Atty. Dkt. No. 088305/0140

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Matthew ROZEK et al.

Title: SYSTEM AND METHOD FOR TRANSFORMING DOCUMENTS TO AND FROM AN XML FORMAT

Appl. No.: 10/026,773

Filing Date: 12/27/2001

Examiner: Laurie Anne Ries

Art Unit: 2176

DECLARATION UNDER 37 C.F.R. § 1.131

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

I, Matthew Rozek, declare as follows:

(1) I am one of the co-inventors of the invention claimed in the captioned application.

(2) Prior to June 21, 2001, the effective prior art date of U.S. Patent No. 6,799,184 to Bhatt et al. [hereinafter "Bhatt"], my co-inventors and I conceived the subject matter of the claimed invention, and subsequent to conception, we diligently reduced it to practice through the development of a working software application embodying the claimed invention.

(3) Exhibit 1 documents the date of February 10, 2001 as the date on which a first invention disclosure was submitted to my company's in-house patent counsel, Edward Swift. Exhibits 2 and 3 document the date of February 15, 2001, as the date on which second and third invention disclosures were submitted to Edward Swift. These three invention disclosures describe each of the elements of the claimed invention and provide substantial evidence of conception of the invention prior to the effective prior date of June 21, 2001 for Bhatt.

(4) After conception, my colleagues and I diligently pursued development of a working software application embodying the claimed invention as set forth below.

(5) Exhibit 4 documents that a complete software application ready for marketing was completed on June 28, 2001. As evidenced by the e-mail of Exhibit 4 dated June 28, 2001, AI (Application Integrator) 4.0 had been successfully completed, and DP3 had been passed. Passage of Decision Point 3 (DP3) indicates that quality assurance has been completed, and the software application is ready to be sold. AI 4.0, as completed on June 28, 2001, embodies all of the aspects of the claimed invention. In addition, since AI 4.0 was completed "three weeks earlier than promised," as stated in the e-mail, it is clear that my colleagues and I diligently reduced the invention to practice from the time of conception.

(5) Exhibit 5 is the "XML Plug-In Data Modeler's Guide" [hereinafter "XML Guide"] for "Application Integrator." Although the date on the first page is "February 2000," this date is erroneous. Our records on our computerized document management system indicate that the date of this document is in fact July 9, 2001. The XML Guide expressly documents in Sections 1 and 2 that Application Integrator generates data models from an XML source, i.e., a DTD, creates mapping rules between a source model and a target model, stores the data models and mapping rules, and verifies or validates that an XML-document is well-formed.

(6) Exhibit 6 is the "XML Schema Plug-In" [hereinafter "XML Schema Guide"] for "Application Integrator" dated October of 2001. The XML Schema Guide expressly documents in the Sections "XML Schema Overview" and "Creating XML Schema Data Models" that Application Integrator generates data models from an XML source, i.e., an XSD, creates mapping rules between a source model and a target model, stores the data models and mapping rules, and verifies or validates that an XML-document is well-formed.

(7) Exhibits 4-6 therefore demonstrate that, after conception of the invention, my colleagues and I diligently developed and reduced to practice a working software application, AI 4.0, which embodies the claimed invention. In addition, the above statements and documentation (including Exhibits 1-6) clearly show evidence of conception prior to the Bhatt effective filing date of June 21, 2001, and continuous diligence from a time prior to the Bhatt effective filing date of June 21, 2001, to the actual reduction to practice of the invention

on June 28, 2001. Accordingly, the present application is entitled to a priority date prior to the Bhatt effective filing date of June 21, 2001.

(10) All of the recited events took place in the United States.

(11) I further declare that all statements made herein of our own knowledge are true, and that all statements made on information and belief are believed to be true; and further that these statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code, and that such willful false statements and the like so made may jeopardize the validity of this declaration, the subject application, or any patent issuing thereon.

Full name: Matthew ROZEK

Signature: 

Date: 7-7-2006

INVENTION DISCLOSURE LETTER FORM

TO: Edward Swift
Intellectual Property Counsel

cc: Joseph P. Dupree
VP, RMS Electronic Commerce Systems

FROM: Matt Rozek

1.0 DESCRIPTION OF THE INVENTION

Please prepare a written description of your invention, based on the outline provided below:

1.1 Title: Provide a title that describes your invention.

"XML Auto-Magic Wizard within Application Integrator(AI)".

1.2 Statement of the Problem: What need does your invention address? Briefly describe the problem or requirement addressed by your invention.

When companies' exchange documents electronically, documents are formatted in a variety of different formats, and usually none of which are in the format ready to be input by the intended destination application. To put these documents into the required application format, a translation process is utilized (translator, such as Application Integrator/OmniTrans). The AI translator uses definitions of the transformation process called data models. These models define both the structures of the input and output, and the rules for moving/manipulating the input data to the output.

Data models representing XML syntax are considerably more complex than data models representing most other public standards. This complexity comes from using a larger number of data model items and more hierarchical levels to properly represent and process the XML syntax. Working with these data model structures requires significantly more User effort to define an environment with mapping rules.

Specifically this invention greatly reduces User effort by generating a testing translation environment for an XML implementation. The complete AI environment is generated, so that a User can immediately execute the translation and receive output without having to do anything else. This generated environment could be used as is, or most likely will be User modified to better match their implementation needs. Users will find that being able to modify an existing environment rather than creating one from scratch significantly reduces their effort and time.

1.3 State of The Art: How has the need been addressed before? Briefly describe how the problem was addressed prior to your invention. Indicate if a literature search has been conducted and list any relevant literature or patents of which you are aware.

Before the invention of Auto-Magic Wizard, the User would run the XML data model generator, which would create a data model containing the structure without any mapping rules. All of the other pieces necessary for a translation environment would be built manually. This includes building the opposite side data model (source or target), adding mapping rules to both data models to move the data, a map component file to configure the environment, a run file to invoke the translation, and an input file to test the translation. Creating these other environment components

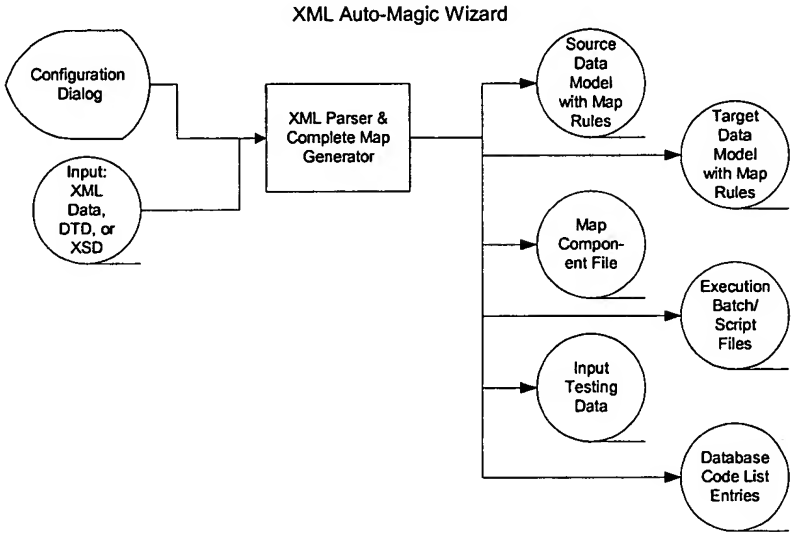
and adding mapping rules to the generated XML data model requires considerable User time and effort.

At this time no search has been done among other translator products to determine if they are able to generate a complete XML translation environment from an XML source.

1.4 **Description and Operation:** How does your invention work? Briefly describe the important features of your invention and explain how to use the invention to solve the problem. Attach any pictures, e.g., drawings, graph, flow charts, photographs, that help to clearly describe the invention (label the pictures with reference numerals and refer to the numerals in your written description).

Translation Environment Wizard

This wizard further minimizes the map development effort. It builds upon the DTD and XSD data model generators. Not only does it generate a data model representing the XML structure, but it generates a fully operational testing environment that allows the user to immediately perform translation, and later customize for their specific implementation needs.



The Wizard has two input, the XML structure definition and a GUI configuration dialog. For XML structure definition, raw XML data, XML DTD and XML XSD schema will be used. With raw XML data, the structure will be extrapolated, whereas with DTD and schema the structure is described.

The configuration dialog is invoked from within Workbench. Here the user identifies the direction of the translation to be created (into or out of XML) and whether the complete environment is to be created or specific part of.

The Wizard will then invoke a parser to read in the XML data (raw XML and XSD schema) or the DTD, and create the various components used in a translation environment. Everything can be generated from this input data:

- The source and target data model defining the XML structure and a mirrored flat file type structure. Within the generated models will be all the rules to move the data from the source to the target, along with loop control logic for properly mapping repeating information.
- Testing input data to drive the generated testing environment is created. This data will be either XML data or in the flat file format, depending on the direction (into or out of XML).

- A file containing lists of codes that will be reference by rules within the generated data models. These codes are defined in DTDs and XSD schemas.
- A map component file which describes the components of the environment to the translator. It defines the input, output, and source and target models.
- A run file (batch, shell script) which allows the user to invoke the translation for this specific implementation.

1.5 **Results:** What advantages are provided by your invention? Briefly describe any efforts to make a prototype of your invention or to test your invention. Summarize the results of any experiments and testing and highlight any results of particular significance.

The advantage of this invention is to be able to quickly and easily create an XML translation environment that can be modified to meet a Users specific implementation needs. Once generated, the User through running test translations can evaluate and modify the components and get immediate feedback on the results. This approach will speed the process from developing XML maps and getting them into a production being used in a business process.

The only parts of this invention that has been prototyped or implemented is the generation of the XML data model from an XML DTD (document type definition). Within this generated data model, structure is defined without mapping rules. Also analysis has identified what the structure of the opposite direction data model will look like, and its data. (The opposite side data model represents the format of the data being translated into XML or from XML.) The creation of this opposite side data model or its data has not been automated yet. Prototyping for any of the other components has not occurred.

1.6 **Execution:** Using the format illustrated below:

(i) Each inventor must sign and date each page of the Description of the Invention and each sheet of any pictures provided as part of the disclosure.

(ii) Two witnesses must read, understand, sign and date each page of the Description of the Invention and each sheet of any pictures provided as part of the disclosure.

Submitted By:		Witnessed, read and understood by:	
_____	_____	_____	_____
Inventor's signature	Date	Witness's Signature	Date
_____	_____	_____	_____
Inventor's signature	Date	Witness's Signature	Date

1.7 **Marking:** Please mark each page of the disclosure with the legend:

GE CONFIDENTIAL - For GE Use Only

GE CONFIDENTIAL - For GE Use Only

2.0 OTHER INFORMATION

2.1 Product line

Indicate the product line, program and/or project to which your invention pertains:

Application Integrator XML Plug-in version 2.0, Application Integrator version 4.0 and future releases.

2.2 Government Contracts

With respect to this invention:	Yes	No
(a) Was the invention conceived, built or tested under a U.S. Government contract?	( )	(X)

(b) If you have answered "YES" to question 2.2 (a) above, please provide the following information and have your manager or finance representative, as appropriate for your business component, verify the information and sign below:

	Conceived	First Made	Teste
Charge Number			
Contract Number			
Date			

I concur with the Contract information indicated above.

Signature	Name	Title	Date

2.3 Disclosure and Use of the Invention

With respect to this invention:	Yes	No
(a) Did anyone other than employees of GE participate in the work that led to the invention (e.g., as part of a joint development project with a third party)?	( )	(X)
(b) Has the invention been disclosed to anyone other than employees of GE (e.g., to a vendor or a potential customer)?	( )	(X)
(c) Has the invention been described in a printed or electronic publication?	(X)	( )
(d) Has the invention been demonstrated for or used by anyone other than employees of GE?	( )	(X)
(e) Has the invention (including both products and processes) been used commercially, either outside GE or within GE's own operations?	( )	(X)
(f) Has the invention been sold or offered for sale?	( )	(X)
(g) Is any disclosure, demonstration, commercial use or offer for sale of the invention planned?	(X)	( )

(h) If you have answered "YES" to any of questions 2.3 (a) - (g), please provide detailed information below regarding the relevant dates and circumstances. Use additional sheets if necessary.

The invention has been described in an internal confidential Requirements Definition Document and a NPI DP0 power point presentation.

We are preparing for development of the invention in the AI 4.0 release, which is expected to be available for beta and later commercial, the summer of 2001.

2.4 Inventor Information

Provide the following information for each inventor. (Add additional pages if needed)

	Inventor 1	Inventor 2	Inventor 3
Full Name			
Current Work Address	RMS Electronic Commerce Systems, Inc. 34405 W. 12 Mile Road Farmington Hills, MI 48331		
Work Phone Number	(248) 324-1242 x131		
Current Home Address			
Citizenship			
Social Security Number			

1. Description of the Invention: This portion of the Invention Disclosure Letter form provides an outline for describing your invention in a "problem-solution" format. In order for the Invention disclosure letter to fulfill its function as evidence of an invention having been made as of the date of the disclosure letter, is important to have each page of the Description of the Invention and each sheet of any pictures provided as part of the disclosure signed by the inventors and witnessed.

2. Other Information: This portion of the Invention Disclosure Letter form requests information regarding Product Line, Government Contract, Disclosure and Use of the Invention and Inventor Information.

2.1 Product Line: The Product Line portion of the Invention Disclosure Letter requests information regarding the product line, program and/or project to which your invention pertains the product line, program and/or project to which the invention pertains.

2.2 Reporting Inventions Under Government Contracts: The Government Contract portion of the Disclosure Letter form requests information regarding the involvement of the U.S. Government in the funding of the work leading to conception of the invention or of any work relating to the first reduction to practice of the invention.

Interim reports are required under government contracts at least every twelve months and must include a listing of inventions conceived or first actually reduced to practice during the period covered by the report. The disclosure must be sufficiently complete to convey a clear understanding of the nature, purpose, operation and important characteristics of the invention. The government can withhold payment of a portion of the payment due under the contract if the required written invention disclosures are not submitted on a timely basis.

2.3 Disclosure and Use of the Invention: The Disclosure and Use portion of the Invention Disclosure Letter format requests information regarding certain events that may have relevance to the patentability of the invention.

Important Note: Property rights in a new invention can be lost if the invention is disclosed outside GE or used commercially prior to filing a patent application directed to the invention. Consult GE Intellectual Property Counsel before making any disclosure of information regarding new GE technology outside GE or making commercial use of new GE technology.

2.4 Inventor Information: The Inventor Information portion of the Invention Disclosure Letter requests information regarding the inventors. It should be noted that with respect to the Invention Disclosure Letter, the term "inventors" is not used in a rigorous sense.

a. A person that makes an independent contribution to the conception of a particular invention is an inventor of that invention. Inventions can be made by a sole inventor acting independently or by a group of coinventors, each of which makes an independent contribution to the conception of the invention, acting jointly. Someone acting solely as "a pair of hands", i.e., that merely carries out the instructions of an inventor without providing any contribution to the conception of the invention, would not be considered a coinventor of the invention.

b. For purpose of submitting an invention disclosure, identify those individuals that appear likely to satisfy the criteria for inventorship or coinventorship. Ultimately, identification of the proper inventors of an invention claimed in a patent application is a legal determination that can only be made based on the scope of the claims of the patent application.

PROCESSING OF INVENTION DISCLOSURES

1. Preliminary Review

1.1 When the Invention Disclosure Letter is received by the patent operation, the disclosure is assigned a "docket" number for tracking purposes, responsibility for the disclosure letter is assigned to a patent attorney, receipt of the disclosure letter is acknowledged and the inventors are notified of the docket number and the name of the attorney to whom responsibility for the disclosure letter has been assigned

1.2 The attorney will conduct a preliminary review of the disclosure to determine, based on the dates of any reported disclosure of the invention outside GE or any commercial activity relating to the invention, if any immediate action is required to protect GE's rights in the invention.

2. Decision-Making

2.1 If immediate action is required, the attorney will contact the technology manager to make an expedited review and decision regarding handling of the disclosure. Expedited review and decision may be triggered, e.g., by a "YES" answer to one of the questions in the "Disclosure and Use " portion of the Invention Disclosure Letter.

2.2 Typically, the invention disclosure will be reviewed by a cross-functional patent committee (typically including representatives of the technical, business and legal functions) at a regularly scheduled meeting and a decision will be made regarding handling of the disclosure.

2.3 Major factors considered in deciding how best to protect the invention:

a. technical significance of the invention;

b. commercial significance of the invention;

c. GE's intention to practice the invention;

d. viability of protecting the invention as a trade secret;

e. competitive advantage lost if GE is excluded from practicing the invention;

f. competitive advantage gained by excluding competitors from practicing invention; and

g. whether invention has been reduced to practice.

2.4 A decision is made whether to:

a. prepare and file a patent application directed to invention;

b. prepare and file a defensive publication that discloses the invention;

c. inactivate the disclosure; and

d. place disclosure on hold for review at next patent committee meeting.

2.5 **Important**: The subject matter of pending patent applications, unpublished non-U.S. patent applications, unactivated invention disclosures and invention disclosures that have been placed on hold should be treated as GE proprietary information.

3. "Session P"

3.1 Ideally, invention disclosures rated for filing as patent applications will be prioritized in a Session P review. Session P is a mechanism for preparing, communicating and executing a competitive, global and cost effective patent strategy in line with business objectives. Representatives of business, technical and IP law functions meet to:

a. Define business objectives;

b. Define technology programs;

c. Evaluate invention disclosures on basis of urgency and financial impact;

d. Define budgetary constraints;

e. Conduct Competitive Analysis; and

f. Conduct GE Portfolio analysis.

INVENTION DISCLOSURE LETTER FORM

TO: Catherine Horrigan
Intellectual Property Counsel
cc: Joseph P. Dupree
VP, RMS Electronic Commerce Systems

FROM: Matthew Rozek
Jamin Williams
James Jeannette

1.0 DESCRIPTION OF THE INVENTION

Please prepare a written description of your invention, based on the outline provided below:

1.1 Title: Provide a title that describes your invention.

"A software program which reads in an XSD (XML Schema Definition) schema and generates source and/or target data models with a code list file for use within Application Integrator (AI)".

1.2 Statement of the Problem: What need does your invention address? Briefly describe the problem or requirement addressed by your invention.

When companies purchase ERP or DRP systems, or connect their software systems such as order processing, shipping, and billing to the Internet, there is a need to quickly translate data transactions from one format to another. Writing and maintaining specific software programs for each translation need is costly. Data transformation products such as RMS Electronic Commerce Systems' Application Integrator (AI), help businesses address their data transformation needs.

When creating maps (aka models) to instruct AI how to transform data to/from specific source and target formats, several days of effort are required. While this is preferable to writing software programs to handle the effort, speed in writing the maps can still be improved.

The XSD schema can be used to define both the source or target's format. The invention (software program) takes the XSD schema and produces AI data models containing the complete data structure and compliance rules, ready for mapping rules to be added. Without this software program, the mapper would have to print a listing of the implementation definition and then manually enter the structure into AI's Workbench tool. This saves a great deal of time for the mapper.

1.3 State of The Art: How has the need been addressed before? Briefly describe how the problem was addressed prior to your invention. Indicate if a literature search has been conducted and list any relevant literature or patents of which you are aware.

Before the invention of this software program (invention), there was no known existing way other than to manually create the data models off of a XSD listing. However, since both the XSD format and the AI product from RMS are available in the market, it is possible that some third party may have or be in the process of creating such a program.

At this time, we are not aware of any company currently marketing or in the process of developing this invention.

1.4 Description and Operation: How does your invention work? Briefly describe the important features of your invention and explain how to use the invention to solve the problem. Attach any

pictures, e.g., drawings, graph, flow charts, photographs, that help to clearly describe the invention (label the pictures with reference numerals and refer to the numerals in your written description).

The process of generating the data models and the code list file is as follows:

- A XSD schema is obtained for the specific message that needs to be modeled and translated within Application Integrator (1).
- The software program (invention) is then executed. It reads the input XSD schema and produced either a source or target data model as specified by the User (2).
- The generated data model can immediately be utilized with AI Workbench to add the necessary mapping rules (3) and output a data transformation map (4).
- The generated code list file can immediately be loaded into the OT Profile Database.
- Translation can then be performed reading in data (5), referencing the XSD schema data model (6) and outputting transformed data (7).

1.5 **Results:** What advantages are provided by your invention? Briefly describe any efforts to make a prototype of your invention or to test your invention. Summarize the results of any experiments and testing and highlight any results of particular significance.

This specific software program (invention) has not yet been prototyped or developed. Two similar software program products have been developed and are commercially available by GE-GXS with the AI product. These products are:

- SEF data model and code list generator – generates data models and a code list file from a Standards Exchange Format (SEF) file that describes the format of a business document.
- DTD data model generator – generates data models from an XML Document Type Definition (DTD) file that describes the format of a business document.

A third product to generate data models is used internally. This internal model generator reads the Washington Publishing format for ASC X12 and UN / EDIFACT business documents and generates both source and target data model templates. These templates are then distributed with AI for these standards.

Using this software program (invention), modelers will be able to implement the Trading Partner implementations that are specified in the XSD schema format in much less time. The software program (invention) generates the data models consistent with those generated by other means. Being consistent, modelers who are experienced with these other generated data models will realize further time savings, since they will take less time to understand the generated data model and less time to test the model for deviation from the XSD specifications.

1.6 **Execution:** Using the format illustrated below:

(i) Each inventor must sign and date <u>each page</u> of the Description of the Invention and each sheet of any pictures provided as part of the disclosure.

(ii) Two witnesses must read, understand, sign and date <u>each page</u> of the Description of the Invention and each sheet of any pictures provided as part of the disclosure.

Submitted By:		Witnessed, read and understood by:	
_____	_____	_____	_____
Inventor's signature	Date	Witness's Signature	Date
_____	_____	_____	_____
Inventor's signature	Date	Witness's Signature	Date

1.7 Marking: Please mark each page of the disclosure with the legend:

GE CONFIDENTIAL - For GE Use Only

GE CONFIDENTIAL - For GE Use Only

2.0 OTHER INFORMATION

2.1 Product line

Indicate the product line, program and/or project to which your invention pertains:

Application Integrator and OmniTrans versions 3.0, 3.1 and future releases.

2.2 Government Contracts

With respect to this invention:	Yes	No
(a) Was the invention conceived, built or tested under a U.S. Government contract?	( )	(X)

(b) If you have answered "YES" to question 2.2 (a) above, please provide the following information and have your manager or finance representative, as appropriate for your business component, verify the information and sign below:

	Conceived	First Made	Teste
Charge Number			
Contract Number			
Date			

I concur with the Contract information indicated above.

Signature	Name	Title	Date

2.3 Disclosure and Use of the Invention

With respect to this invention:	Yes	No
(a) Did anyone other than employees of GE participate in the work that led to the invention (e.g., as part of a joint development project with a third party)?	( )	(X)
(b) Has the invention been disclosed to anyone other than employees of GE (e.g., to a vendor or a potential customer)?	( )	(X)
(c) Has the invention been described in a printed or electronic publication?	(X)	( )
(d) Has the invention been demonstrated for or used by anyone other than employees of GE?	( )	(X)
(e) Has the invention (including both products and processes) been used commercially, either outside GE or within GE's own operations?	( )	(X)
(f) Has the invention been sold or offered for sale?	( )	(X)
(g) Is any disclosure, demonstration, commercial use or offer for sale of the invention planned?	(X)	( )

(h) If you have answered "YES" to any of questions 2.3 (a) - (g), please provide detailed information below regarding the relevant dates and circumstances. Use additional sheets if necessary.

The invention has been described in an internal confidential Requirements Definition Document and a NPI DP0 power point presentation.

We are preparing for development of the invention in the XML Plug-in version 2.0, which is expected to be available for beta and later commercial, the spring of 2001.

2.4 Inventor Information

Provide the following information for each inventor. (Add additional pages if needed)

	Inventor 1	Inventor 2	Inventor 3
Full Name	Matt Rozek	Jamin Williams	James Jeannette
Current Work Address	RMS Electronic Commerce Systems, Inc. 34405 W. 12 Mile Road Farmington Hills, MI 48331	RMS Electronic Commerce Systems, Inc. 34405 W. 12 Mile Road Farmington Hills, MI 48331	RMS Electronic Commerce Systems, Inc. 34405 W. 12 Mile Road Farmington Hills, MI 48331
Work Phone Number	(248) 324-1242 x110	(248) 324-1242 x131	(248) 324-1242 x115
Current Home Address	25315 Sullivan Lane Novi, MI 48375-1419	14203 Springlake Blvd. Novi, MI 48377	2786 Ripple Way White Lake, MI 48383
Citizenship	United States	United States	United States
Social Security Number	371-54-6072	092-54-4310	

1. Description of the Invention: This portion of the Invention Disclosure Letter form provides an outline for describing your invention in a "problem-solution" format. In order for the Invention disclosure letter to fulfill its function as evidence of an invention having been made as of the date of the disclosure letter, is important to have each page of the Description of the Invention and each sheet of any pictures provided as part of the disclosure signed by the inventors and witnessed.

2. Other Information: This portion of the Invention Disclosure Letter form requests information regarding Product Line, Government Contract, Disclosure and Use of the Invention and Inventor Information.

2.1 Product Line: The Product Line portion of the Invention Disclosure Letter requests information regarding the product line, program and/or project to which your invention pertains the product line, program and/or project to which the invention pertains.

2.2 Reporting Inventions Under Government Contracts: The Government Contract portion of the Disclosure Letter form requests information regarding the involvement of the U.S. Government in the funding of the work leading to conception of the invention or of any work relating to the first reduction to practice of the invention.

Interim reports are required under government contracts at least every twelve months and must include a listing of inventions conceived or first actually reduced to practice during the period covered by the report. The disclosure must be sufficiently complete to convey a clear understanding of the nature, purpose, operation and important characteristics of the invention. The government can withhold payment of a portion of the payment due under the contract if the required written invention disclosures are not submitted on a timely basis.

2.3 Disclosure and Use of the Invention: The Disclosure and Use portion of the Invention Disclosure Letter format requests information regarding certain events that may have relevance to the patentability of the invention.

Important Note: Property rights in a new invention can be lost if the invention is disclosed outside GE or used commercially prior to filing a patent application directed to the invention. Consult GE Intellectual Property Counsel before making any disclosure of information regarding new GE technology outside GE or making commercial use of new GE technology.

2.4 Inventor Information: The Inventor Information portion of the Invention Disclosure Letter requests information regarding the inventors. It should be noted that with respect to the Invention Disclosure Letter, the term "inventors" is not used in a rigorous sense.

a. A person that makes an independent contribution to the conception of a particular invention is an inventor of that invention. Inventions can be made by a sole inventor acting independently or by a group of coinventors, each of which makes an independent contribution to the conception of the invention, acting jointly. Someone acting solely as "a pair of hands", i.e., that merely carries out the instructions of an inventor without providing any contribution to the conception of the invention, would not be considered a coinventor of the invention.

b. For purpose of submitting an invention disclosure, identify those individuals that appear likely to satisfy the criteria for inventorship or coinventorship. Ultimately, identification of the proper inventors of an invention claimed in a patent application is a legal determination that can only be made based on the scope of the claims of the patent application.

PROCESSING OF INVENTION DISCLOSURES

1. Preliminary Review

1.1 When the Invention Disclosure Letter is received by the patent operation, the disclosure is assigned a "docket" number for tracking purposes, responsibility for the disclosure letter is assigned to a patent attorney, receipt of the disclosure letter is acknowledged and the inventors are notified of the docket number and the name of the attorney to whom responsibility for the disclosure letter has been assigned

1.2 The attorney will conduct a preliminary review of the disclosure to determine, based on the dates of any reported disclosure of the invention outside GE or any commercial activity relating to the invention, if any immediate action is required to protect GE's rights in the invention.

2. Decision-Making

2.1 If immediate action is required, the attorney will contact the technology manager to make an expedited review and decision regarding handling of the disclosure. Expedited review and decision may be triggered, e.g., by a "YES" answer to one of the questions in the "Disclosure and Use " portion of the Invention Disclosure Letter.

2.2 Typically, the invention disclosure will be reviewed by a cross-functional patent committee (typically including representatives of the technical, business and legal functions) at a regularly scheduled meeting and a decision will be made regarding handling of the disclosure.

2.3 Major factors considered in deciding how best to protect the invention:

a. technical significance of the invention;

b. commercial significance of the invention;

c. GE's intention to practice the invention;

d. viability of protecting the invention as a trade secret;

e. competitive advantage lost if GE is excluded from practicing the invention;

f. competitive advantage gained by excluding competitors from practicing invention; and

g. whether invention has been reduced to practice.

2.4 A decision is made whether to:

a. prepare and file a patent application directed to invention;

b. prepare and file a defensive publication that discloses the invention;

c. inactivate the disclosure; and

d. place disclosure on hold for review at next patent committee meeting.

2.5 **Important**: The subject matter of pending patent applications, unpublished non-U.S. patent applications, unactivated invention disclosures and invention disclosures that have been placed on hold should be treated as GE proprietary information.

3. "Session P"

3.1 Ideally, invention disclosures rated for filing as patent applications will be prioritized in a Session P review. Session P is a mechanism for preparing, communicating and executing a competitive, global and cost effective patent strategy in line with business objectives. Representatives of business, technical and IP law functions meet to:

a. Define business objectives;

b. Define technology programs;

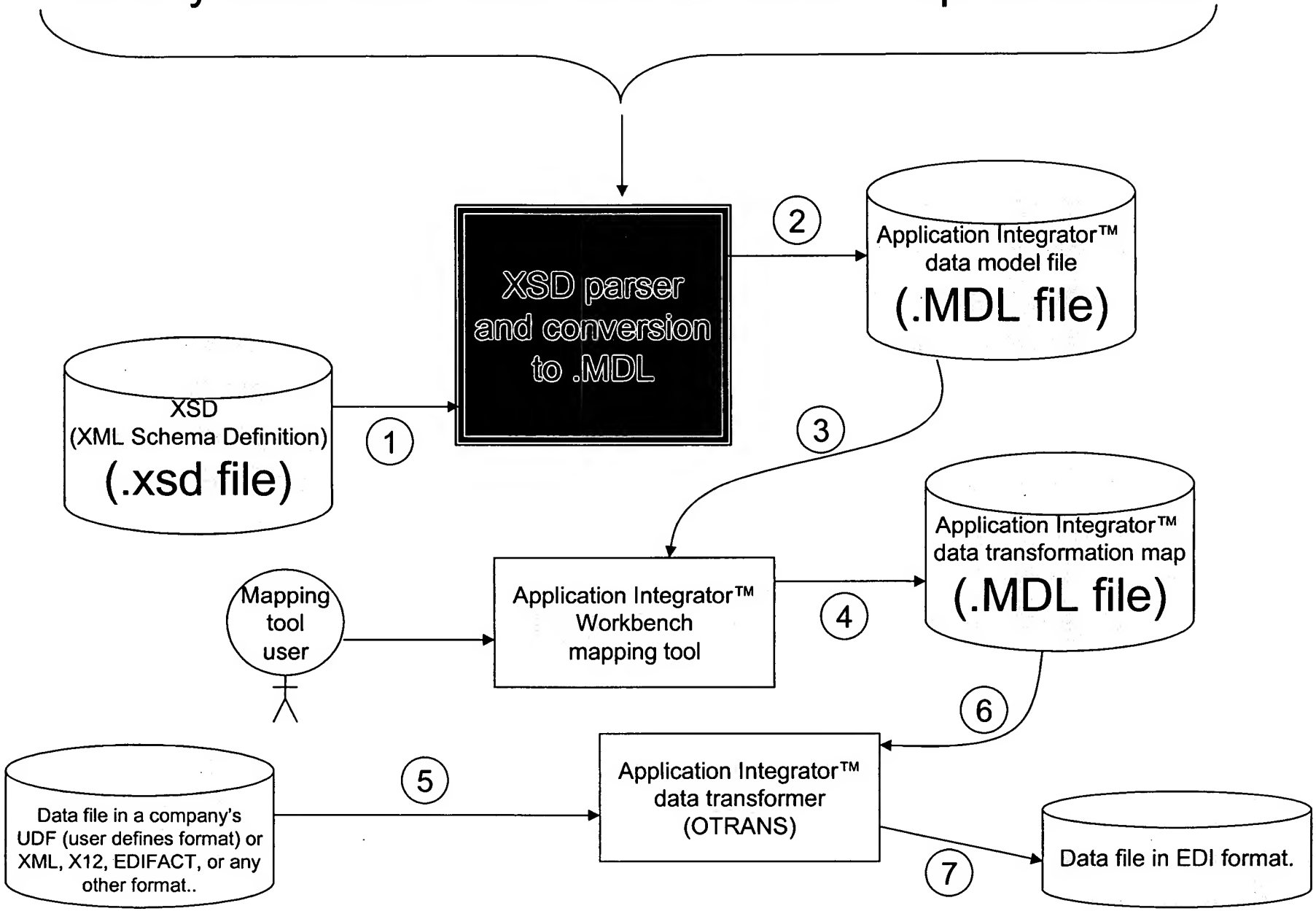
c. Evaluate invention disclosures on basis of urgency and financial impact;

d. Define budgetary constraints;

e. Conduct Competitive Analysis; and

f. Conduct GE Portfolio analysis.

Partially automated data-transformation map constructor.



INVENTION DISCLOSURE LETTER FORM

TO: Catherine Horrigan
Intellectual Property Counsel
cc: Joseph P. Dupree
VP, RMS Electronic Commerce Systems

FROM: Matthew Rozek
Jamin Williams
James Jeannette

1.0 DESCRIPTION OF THE INVENTION

Please prepare a written description of your invention, based on the outline provided below:

1.1 Title: Provide a title that describes your invention.

"A software process within Application Integrator (AI), which transforms messages either into or out of an XSD (XML Schema Definition) schema format".

1.2 Statement of the Problem: What need does your invention address? Briefly describe the problem or requirement addressed by your invention.

When companies purchase ERP or DRP systems, or connect their software systems such as order processing, shipping, and billing to the Internet, there is a need to quickly translate data transactions from one format to another. Writing and maintaining specific software programs for each translation need is costly. Data transformation products such as RMS Electronic Commerce Systems' Application Integrator (AI) help businesses address their data transformation needs.

This invention specifically addresses the need to translate data either from or to an XML format, based on XSD schema. This process addresses the need to not only check that the data is well formed, but also to validate the XML data against a given XSD schema. Without this checking and validation occurring, the data being processed may be corrupt and therefore when translated, the output would be inaccurate or erroneous.

1.3 State of The Art: How has the need been addressed before? Briefly describe how the problem was addressed prior to your invention. Indicate if a literature search has been conducted and list any relevant literature or patents of which you are aware.

Before the invention of this process, XML data, based on XSD schema, can be processed, but only without validation occurring. The input XML data would not be compared to the XSD schema to insure that it complies with this structure definition. The data can be checked to follow basic XML syntax rules, but not checked to be in compliance with the message's specific requirements. Examples of some of these requirements would be:

- Field requirement – mandatory, optional or conditional on other fields
- Field minimum and maximum size
- Field character set
- Field range or pattern of values

At this time, we are not aware of any company currently marketing or in the process of developing this invention to work with Application Integrator.

1.4 **Description and Operation:** How does your invention work? Briefly describe the important features of your invention and explain how to use the invention to solve the problem. Attach any pictures, e.g., drawings, graph, flow charts, photographs, that help to clearly describe the invention (label the pictures with reference numerals and refer to the numerals in your written description).

During the inbound translation of XML data based on XSD schema, the AI translator invokes the XSD parser routine (1). This routine then reads in the XML data (2) and identifies by reference within the XML, the XSD schema which will be used to validate the XML data. The parser reads in the schema (3) and returns back to reading in the XML data, while checking the data to be well formed and validating it against the schema. As portions of the data pass the parser's checking, the data is made available to the AI Translator (4), which reformats and outputs the data into another format(5).

With outbound translation into the XML format based on XSD schema, the AI translator reads in the data to be transformed into XML data (1). AI translates the data and passes its output to the XSD parser routine (2). The XSD parser routine identifies within the data which schema to validate against, and reads in the XSD schema (3). As the parser checks the data to be well formed, it also validates against the schema and outputs the XML data (4).

1.5 **Results:** What advantages are provided by your invention? Briefly describe any efforts to make a prototype of your invention or to test your invention. Summarize the results of any experiments and testing and highlight any results of particular significance.

This specific software program (invention) has not yet been prototyped or developed. A similar software program product has been developed and is commercially available by GE-GXS with the AI product. This product is the XML Plug-in version 1.0, which performs AI translations using DTDs. This process checks the XML data to be well formed and validates it against a Document Type Definition (DTD).

Using this software program (invention), XSD schema based XML data can also be validated during translation. Without it, validation will not take place to insure that the XML data is in full compliance with its message definition. By not insuring its in full compliance, erroneous or corrupt data could be unknowingly accepted or output by the translation process.

1.6 **Execution:** Using the format illustrated below:

(i) Each inventor must sign and date each page of the Description of the Invention and each sheet of any pictures provided as part of the disclosure.

(ii) Two witnesses must read, understand, sign and date each page of the Description of the Invention and each sheet of any pictures provided as part of the disclosure.

Submitted By:		Witnessed, read and understood by:	
_____	_____	_____	_____
Inventor's signature	Date	Witness's Signature	Date
_____	_____	_____	_____
Inventor's signature	Date	Witness's Signature	Date

1.7 **Marking:** Please mark each page of the disclosure with the legend:

GE CONFIDENTIAL - For GE Use Only

GE CONFIDENTIAL - For GE Use Only

2.0 OTHER INFORMATION

2.1 Product line

Indicate the product line, program and/or project to which your invention pertains:

Application Integrator and OmniTrans versions 3.0, 3.1 and future releases.

2.2 Government Contracts

With respect to this invention:	Yes	No
(a) Was the invention conceived, built or tested under a U.S. Government contract?	( )	(X)

(b) If you have answered "YES" to question 2.2 (a) above, please provide the following information and have your manager or finance representative, as appropriate for your business component, verify the information and sign below:

	Conceived	First Made	Teste
Charge Number			
Contract Number			
Date			

I concur with the Contract information indicated above.

Signature	Name	Title	Date

2.3 Disclosure and Use of the Invention

With respect to this invention:	Yes	No
(a) Did anyone other than employees of GE participate in the work that led to the invention (e.g., as part of a joint development project with a third party)?	( )	(X)
(b) Has the invention been disclosed to anyone other than employees of GE (e.g., to a vendor or a potential customer)?	( )	(X)
(c) Has the invention been described in a printed or electronic publication?	(X)	( )
(d) Has the invention been demonstrated for or used by anyone other than employees of GE?	( )	(X)
(e) Has the invention (including both products and processes) been used commercially, either outside GE or within GE's own operations?	( )	(X)
(f) Has the invention been sold or offered for sale?	( )	(X)
(g) Is any disclosure, demonstration, commercial use or offer for sale of the invention planned?	(X)	( )

(h) If you have answered "YES" to any of questions 2.3 (a) - (g), please provide detailed information below regarding the relevant dates and circumstances. Use additional sheets if necessary.

The invention has been described in an internal confidential Requirements Definition Document and a NPI DP0 power point presentation.

We are preparing for development of the invention in the XML Plug-in version 2.0, which is expected to be available for beta and later commercial, the spring of 2001.

2.4 Inventor Information

Provide the following information for each inventor. (Add additional pages if needed)

	Inventor 1	Inventor 2	Inventor 3
Full Name	Matt Rozek	Jamin Williams	James Jeannette
Current Work Address	RMS Electronic Commerce Systems, Inc. 34405 W. 12 Mile Road Farmington Hills, MI 48331	RMS Electronic Commerce Systems, Inc. 34405 W. 12 Mile Road Farmington Hills, MI 48331	RMS Electronic Commerce Systems, Inc. 34405 W. 12 Mile Road Farmington Hills, MI 48331
Work Phone Number	(248) 324-1242 x110	(248) 324-1242 x131	(248) 324-1242 x115
Current Home Address	25315 Sullivan Lane Novi, MI 48375-1419	14203 Springlake Blvd. Novi, MI 48377	2786 Ripple Way White Lake, MI 48383
Citizenship	United States	United States	United States
Social Security Number	371-54-6072	092-54-4310	365-92-1895

1. Description of the Invention: This portion of the Invention Disclosure Letter form provides an outline for describing your invention in a "problem-solution" format. In order for the Invention disclosure letter to fulfill its function as evidence of an invention having been made as of the date of the disclosure letter, is important to have each page of the Description of the Invention and each sheet of any pictures provided as part of the disclosure signed by the inventors and witnessed.

2. Other Information: This portion of the Invention Disclosure Letter form requests information regarding Product Line, Government Contract, Disclosure and Use of the Invention and Inventor Information.

2.1 Product Line: The Product Line portion of the Invention Disclosure Letter requests information regarding the product line, program and/or project to which your invention pertains the product line, program and/or project to which the invention pertains.

2.2 Reporting Inventions Under Government Contracts: The Government Contract portion of the Disclosure Letter form requests information regarding the involvement of the U.S. Government in the funding of the work leading to conception of the invention or of any work relating to the first reduction to practice of the invention.

Interim reports are required under government contracts at least every twelve months and must include a listing of inventions conceived or first actually reduced to practice during the period covered by the report. The disclosure must be sufficiently complete to convey a clear understanding of the nature, purpose, operation and important characteristics of the invention. The government can withhold payment of a portion of the payment due under the contract if the required written invention disclosures are not submitted on a timely basis.

2.3 Disclosure and Use of the Invention: The Disclosure and Use portion of the Invention Disclosure Letter format requests information regarding certain events that may have relevance to the patentability of the invention.

Important Note: Property rights in a new invention can be lost if the invention is disclosed outside GE or used commercially prior to filing a patent application directed to the invention. Consult GE Intellectual Property Counsel before making any disclosure of information regarding new GE technology outside GE or making commercial use of new GE technology.

2.4 Inventor Information: The Inventor Information portion of the Invention Disclosure Letter requests information regarding the inventors. It should be noted that with respect to the Invention Disclosure Letter, the term "inventors" is not used in a rigorous sense.

a. A person that makes an independent contribution to the conception of a particular invention is an inventor of that invention. Inventions can be made by a sole inventor acting independently or by a group of coinventors, each of which makes an independent contribution to the conception of the invention, acting jointly. Someone acting solely as "a pair of hands", i.e., that merely carries out the instructions of an inventor without providing any contribution to the conception of the invention, would not be considered a coinventor of the invention.

b. For purpose of submitting an invention disclosure, identify those individuals that appear likely to satisfy the criteria for inventorship or coinventorship. Ultimately, identification of the proper inventors of an invention claimed in a patent application is a legal determination that can only be made based on the scope of the claims of the patent application.

PROCESSING OF INVENTION DISCLOSURES

1. Preliminary Review

1.1 When the Invention Disclosure Letter is received by the patent operation, the disclosure is assigned a "docket" number for tracking purposes, responsibility for the disclosure letter is assigned to a patent attorney, receipt of the disclosure letter is acknowledged and the inventors are notified of the docket number and the name of the attorney to whom responsibility for the disclosure letter has been assigned

1.2 The attorney will conduct a preliminary review of the disclosure to determine, based on the dates of any reported disclosure of the invention outside GE or any commercial activity relating to the invention, if any immediate action is required to protect GE's rights in the invention.

2. Decision-Making

2.1 If immediate action is required, the attorney will contact the technology manager to make an expedited review and decision regarding handling of the disclosure. Expedited review and decision may be triggered, e.g., by a "YES" answer to one of the questions in the "Disclosure and Use " portion of the Invention Disclosure Letter.

2.2 Typically, the invention disclosure will be reviewed by a cross-functional patent committee (typically including representatives of the technical, business and legal functions) at a regularly scheduled meeting and a decision will be made regarding handling of the disclosure.

2.3 Major factors considered in deciding how best to protect the invention:

a. technical significance of the invention;

b. commercial significance of the invention;

c. GE's intention to practice the invention;

d. viability of protecting the invention as a trade secret;

e. competitive advantage lost if GE is excluded from practicing the invention;

f. competitive advantage gained by excluding competitors from practicing invention; and

g. whether invention has been reduced to practice.

2.4 A decision is made whether to:

a. prepare and file a patent application directed to invention;

b. prepare and file a defensive publication that discloses the invention;

c. inactivate the disclosure; and

d. place disclosure on hold for review at next patent committee meeting.

2.5 **Important**: The subject matter of pending patent applications, unpublished non-U.S. patent applications, unactivated invention disclosures and invention disclosures that have been placed on hold should be treated as GE proprietary information.

3. "Session P"

3.1 Ideally, invention disclosures rated for filing as patent applications will be prioritized in a Session P review. Session P is a mechanism for preparing, communicating and executing a competitive, global and cost effective patent strategy in line with business objectives. Representatives of business, technical and IP law functions meet to:

a. Define business objectives;

b. Define technology programs;

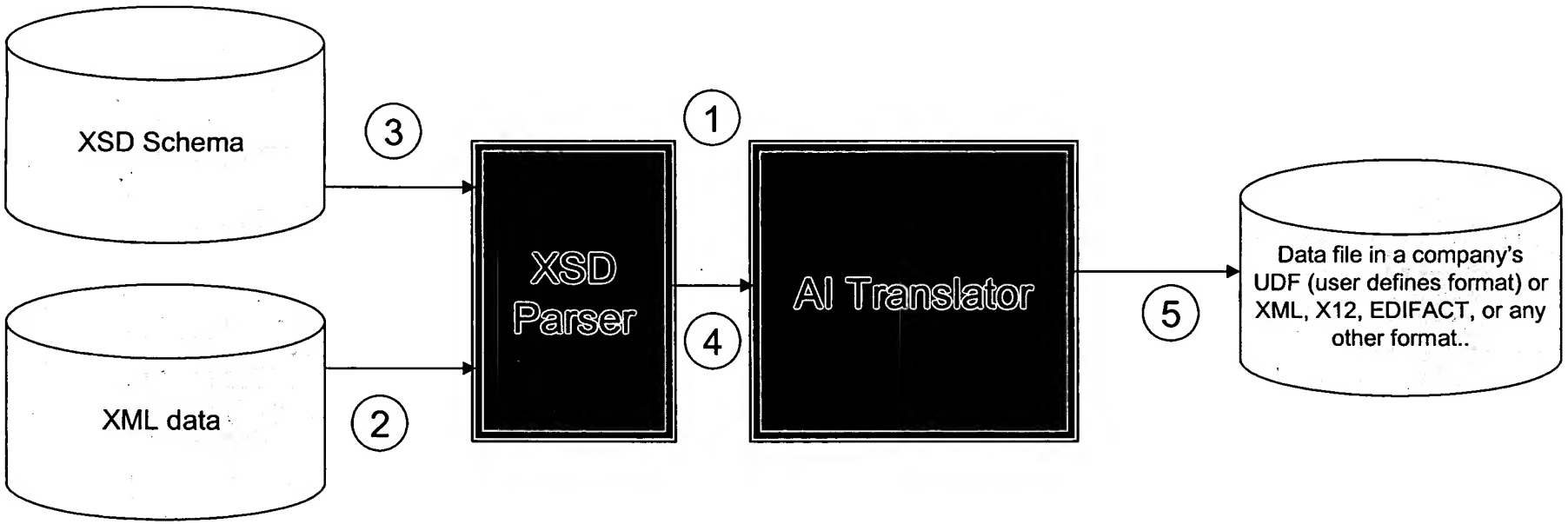
c. Evaluate invention disclosures on basis of urgency and financial impact;

d. Define budgetary constraints;

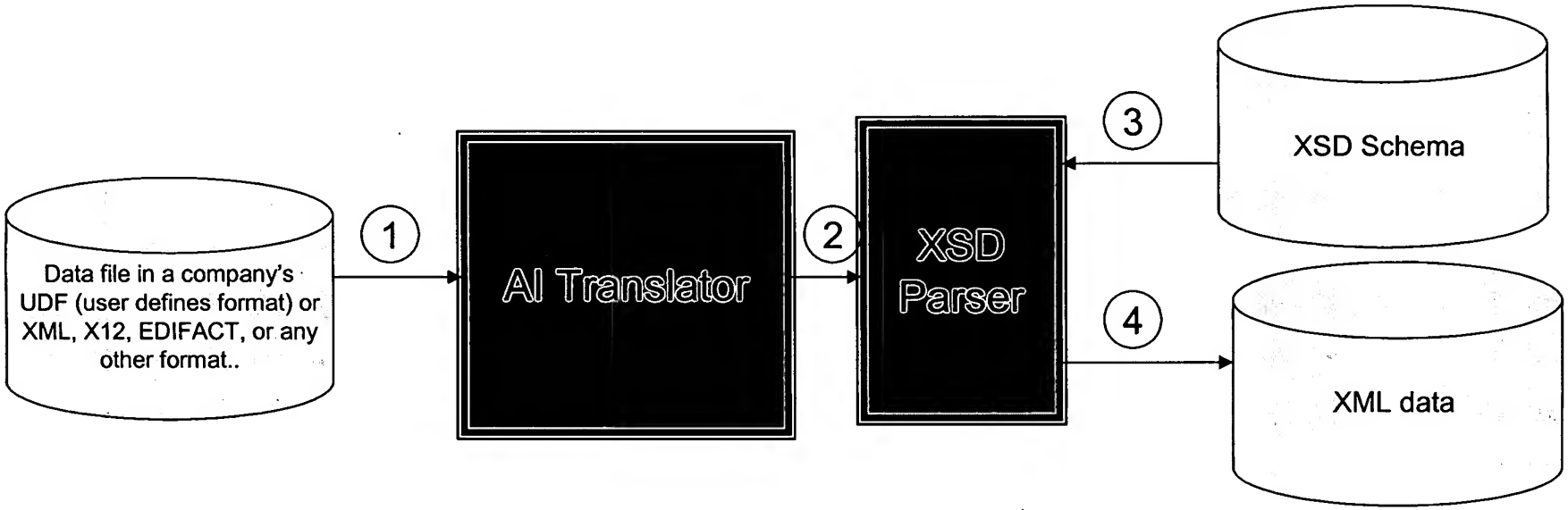
e. Conduct Competitive Analysis; and

f. Conduct GE Portfolio analysis.

Inbound Translation



Outbound Translation



McDaniel, Sharon L.

From: Weinstein, Marc K.
Sent: Thursday, July 06, 2006 10:26 PM
To: 'Edward.Swift@gxs.com'
Subject: Exhibit 4

-----Original Message-----
From: IMCEAEX-_O=GE_OU=GEISROCKVILLE_CN=RECIPIENTS_CN=JOE+2EDUPREE@gxs.com [mailto:IMCEAEX-_O=GE_OU=GEISROCKVILLE_CN=RECIPIENTS_CN=JOE+2EDUPREE@gxs.com]
Sent: Thursday, June 28, 2001 10:16 AM
To: AIENGINEERING@gxs.com
Subject: AI 4.0 DP3 - we did it!

AI Team,

Thanks to *everyone here* who pulled together to make AI 4.0 a success. We passed our DP3 today with an enthusiastic **GO**. We're three weeks earlier than promised, and you should be very proud of your results. The sales and marketing teams are beaming with excitement in having this new release to sell.

Thanks for your hard work and the extra effort you've put into this project.

FYI - a note from Rick Bodson this morning... see below.

Also know that there are a few loose ends that we're still tying up for 4.0... the finalizing the readme.txt and burning the master CDs, etc. Thanks in advance to those putting in the final touches.

Regards,

Joe

-----Original Message-----
From: Bodson, Rick (GXS)
Sent: Thursday, June 28, 2001 9:16 AM
To: Dupree, Joe (GXS); Knauss, Daniel (GXS, RMS)
Subject: I only have one question for the AI team...

The Big Dogs of AI got out in front of the schedule and made it happen! Congratulations to the team for the early delivery of release 4.0!

/Rick

5125.71-19

Application Integrator ™

XML Plug-In Data Modeler's Guide

(Release 1.1)

February 2000

DISCLAIMER

Information in this document is subject to change without notice. GE Information Services reserves the right to change (upgrade) data to provide the most accurate, reliable quality product available. Specific mention of a product in this document is not a guarantee by GE Information Services of complete hardware and software compatibility with your data processing system. If you have questions about hardware and/or software compatibility, please contact your GE Information Services representative.

The following electronic data interchange (EDI) standards are developed and maintained by these organizations:

TRADACOMS	Article Number Association
X12	Accredited Standards Committee (ASC) and Data Interchange Standards Association (DISA), the secretariat and administrative branch for ASC X12
UN/EDIFACT	Data Interchange Standards Association (DISA), the secretariat and publisher for the Pan American EDIFACT Board (PAEB)

TRADEMARKS

Trade Guide, Trade Guide for System Administration, Transaction Modeler Workbench, MapBuilder are trademarks, and RuleBuilder, Interactive Gateway Extension, and User Exit Extension are registered trademarks of RMS Electronic Services, Inc.

c-tree and c-tree Plus are trademarks of Faircom Corporation.

Cleo is a registered trademark of Interface Systems, Inc.

DEC, Digital Alpha Station, and Digital UNIX are trademarks of Digital Equipment Corporation.

eXceed is a trademark of Hummingbird Communications, Inc.

HP, Hewlett-Packard, HP-UX are registered trademarks of Hewlett-Packard.

Intel is a registered trademark of Intel Corporation.

IBM, AIX, Risc System/6000 and RS6000 are trademarks of the International Business Machine, Inc.

SCO is a trademark of the Santa Cruz Operation, Inc.

SPARC is a trademark of SPARC International, Inc.

Sun, Sun Microsystems, and Solaris are registered trademarks of Sun Microsystems, Inc.

XENIX, NT, Windows and Microsoft are registered trademarks and WinSock is a trademark of Microsoft Corporation.

All product names and corporations mentioned in this document may be trademarks, registered trademarks, or copyrighted by their respective owners.

Table of Contents

Preface

The scope of the *Application Integrator XML Plug-In Data Modeler's Guide* is to provide data modelers with the information specific to the XML syntax to develop, test, troubleshoot, and move data models that use XML into a production setting. This manual also provides information about the parser, source and target model examples, and the DTD Data Model Generator.

Any operation dealing with Workbench or testing data models will be found in the *Transaction Modeler Workbench User's Guide* or the *RosettaNet Standards Implementation Guide.* This preface contains information on:

- XML documentation
- Prerequisites for XML
- Documentation conventions
- Mouse and keyboard conventions
- Keyboard shortcuts
- On-line Help
- Application Integrator Customer Support

About the XML Documentation Set

The Application Integrator XML documentation set consists of the following manuals:

Document	Purpose
RosettaNet Standard Implementation Guide	Provides instructions on using Trade Guide with the RosettaNet implementaiton for setting up and runnign translations, and administering the systems. It should be used with the *Trade Guide for System Administration User's Guide.*
AI XML Plug-In Data Modeler's Guide *(this manual)*	Provides instructions and procedures for developing data models using XML syntax. This guide should be used with both the *Transaction Modeler Workbench User's Guide* and the *RosettaNet Standard Implementation Guide.*

About the XML Data Modeler's Guide

This guide is designed to give you a working understanding of the operation and capabilities of the software. The information in this document is arranged so you can quickly and easily understand the features, menus, and operations.

The *XML Data Modeler's Guide* is divided into the following sections and appendixes:

Section	Description
Section 1: XML Overview	Provides an overview of the XML features and terminology, and instructions for installing the product.
Section 2: Creating XML Data Models	Provides complete procedures for creating data models using the data model generator.
Section 3: Testing and Debugging XML Data Models	Provides information about debugging parser and translation errors.
Section 4: XML Examples	Provides an inbound and an outbound example for XML processing.

Section	Description
Section 5: The Data Modeling Process	Provides a discussion and steps to the data modeling process, including Application Integrator conventions and tips.
Appendix A: XML Plug-In Implementation Files	Lists all the files shipped with the XML Plug-In.
Appendix B: XML Item Types	Lists and briefly describes the item types defined by the access models used for XML data modeling.

Prerequisites for using XML

System Prerequisites

For information on the prerequisite hardware and software necessary to run Application Integrator on UNIX systems, including the Workbench and Trade Guide components, refer to the *Application Integrator System Configuration Requirements* document.
For information on the prerequisite hardware and software necessary to run Application Integrator on Windows systems, including the Workbench and Trade Guide components, refer to Section 2 of the *Application Integrator Installation Guide.*

You must have Internet access to allow the XML parser to find nonlocal document type definitions.

Note: We recommend using the Windows default colors for your Application Integrator Windows applications. Changing or customizing your display colors may result in readability problems when using Application Integrator in a Windows environment.

User Prerequisites

It is helpful to have the following background before creating data models that use XML syntax:

- Mouse and graphical user interface (GUI) experience - windows and dialog boxes
- Basic knowledge of your operating system and an on-line editor
- Program concept knowledge, including
  - an understanding of data organization
  - an understanding of data manipulation
  - an understanding of program process flow
  - an understanding of testing and debugging
- Knowledge of electronic data interchange, database management, and systems reporting
- Knowledge of the XML syntax structure

Documentation Conventions

Typographical Conventions

Regular	This text style is used in general.
`Courier`	This text style is used for system output and syntax examples.
Italic	This text style is used for book titles, new terms, and emphasis words.

User Input

In this document, anything printed in Courier and boldface type should be entered exactly as written. For example, if you need to enter the term "userid," it will be shown in the documentation as **`userid`**.

Notes, Hints, and Cautions

Notes provide additional information and are boxed inside the text, using the following format.

Note: This is a note.

Hints provide helpful tips on performing operations in a quicker manner. They are formatted in the same way.

Hint: This is a hint.

Cautions provide information on practices or places where you could possibly overwrite data or program files. They appear in the following format.

Caution: This is a caution.

Screen Images

The screen images in this manual were taken from the Windows version Application Integrator running on a Windows 95 platform. If you are running Application Integrator on Windows NT or UNIX, the actual screens (windows and dialog boxes) may differ slightly in appearance. Differences between platforms are noted throughout this manual where appropriate.

Tables

Tables appear frequently in this manual and are indicated by headings followed by dark underlining then the body of the table without gridlines (in most cases). The end of the table is indicated by double underlines.

Mouse and Keyboard Conventions

In most cases, you can use either a mouse or a keyboard to enter, view, and manipulate the windows and dialog boxes of Application Integrator.

Keyboard Shortcuts

In many instances, you will have the option of either using the mouse or keyboard to perform an action. These keyboard shortcuts are shown to the right of the menu item. For example, to use the Find shortcut you would press and hold the Ctrl key then press the F key. This shortcut is shown as Ctrl+F on the drop down menu and is how they will be listed in this manual.

On-line Help

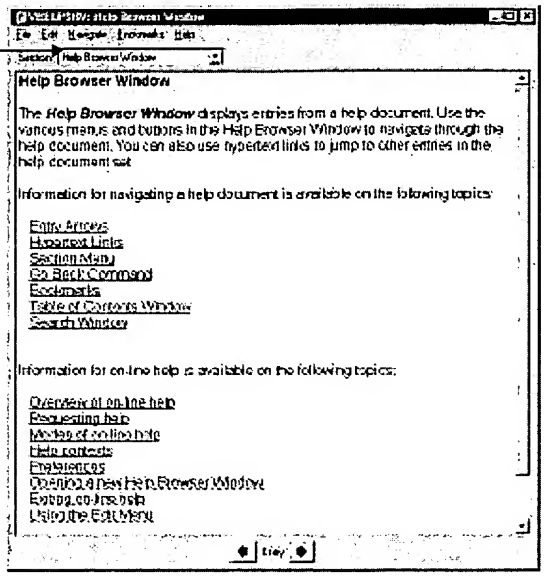
Application Integrator comes with a Help system to assist you on-line while setting up and maintaining translations and administering the system. The Application Integrator on-line Help system opens a Help window with search and navigation features.

To Get Help on Help

1. From the main menu, choose Help. The Help menu appears. From the Help menu, choose Error Code Reference.
2. From the dialog box, choose the Help menu and then choose the Help on the Browser option. You can also press Ctrl+H.

   A Help entry on the Help system itself appears.

Section value entry box



3. Click a hypertext entry (any underlined text) or use the Section value entry box to learn more about on-line Help.
4. To close the Help entry and Help browser, from the File menu, choose Close Browser. You can also press Ctrl+W.

Note: The Section value entry box on the Help window provides a means to quickly access all the major Help menus. Click the indicator (in UNIX – a small rectangle; in Windows – a down arrow) and a list of all major Help menus appear. Click any menu name to move between them.

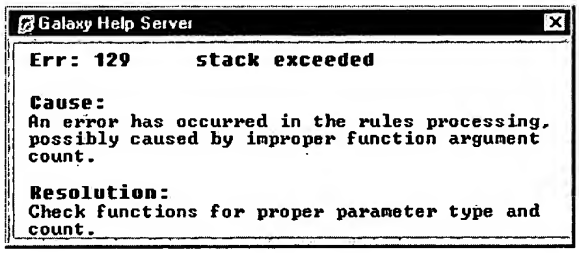
To Use Application Integrator Help

1. From the main menu, choose Help. The Help menu appears.
2. From the Help menu, choose one of the following options:
   - Error Number Reference
   - Keyword/Function Reference
   - About

➢ **If you choose Error Number Reference . . .**

- When a listing of error code *ranges* appears, click the range on which to obtain Help.
- When a listing of error codes appears, click once on the exact error message for assistance on it.

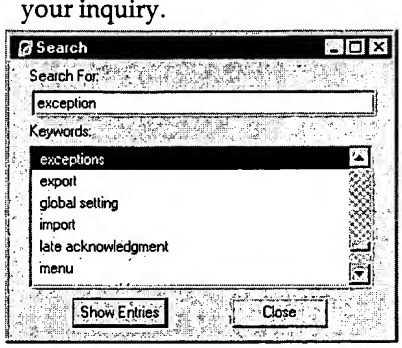
A Help entry, such as the following, appears:



To search for a Help entry

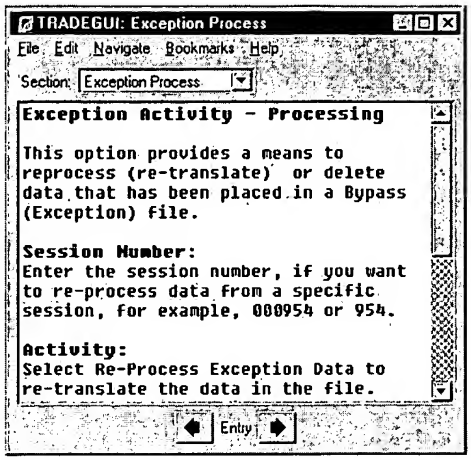
a. To search for a Help entry, from the Navigate menu of any Help entry, choose Search.

b. From the Search window, such as the one below, either scroll to find a keyword matching your inquiry or start typing a keyword and then double-click the keyword in the list box that matches your inquiry.



c. Click the Show Entries button. A Search Navigator window, similar to the following, appears next to the actual Help entry.

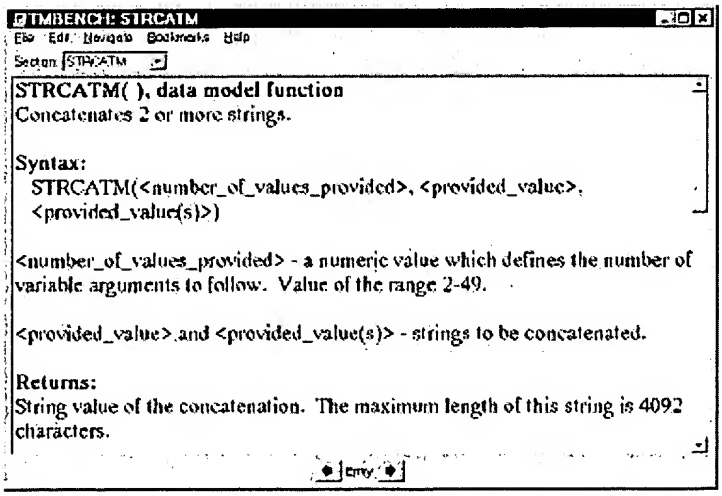
d. Double-click the desired Help entry or highlight the desired Help entry and choose Go To. The appropriate Help entry appears. The following is the actual Help entry for the exception processing option.



➢ **If you choose Keyword/Function Reference**

- ❒ Scroll to the desired function or topic.
- ❒ Click on it.

A Help entry, such as the following appears.



➢ **If you choose About**

A Help entry, such as the one below, appears when you choose the menu selection.

To close the Help entry and Help browser, from the File menu, choose Close Browser. You can also press Ctrl+W.

Application Integrator Customer Support

Application Integrator support is offered through a separate Support Services Agreement. The type of support offered is described within the contract. The support services cover:

- Application Integrator program updates
- Application Integrator model updates (for example, generic and public standards-specific data models)
- Application Integrator standards updates (for example, ASC X12 annual version updates)
- Help Desk Support

Calling for Customer Support

To more effectively help you when you call in for support, follow these steps:

1. If possible, attempt to resolve the question internally or through the Application Integrator documentation, including the print, on-line, and training documentation.
2. Make sure you have copied down the exact Application Integrator version for which you are seeking assistance. The version is found by selecting the About option on the Help menu of Application Integrator. Also copy down the compile version date of the Control Server (called *cservr* in UNIX and *cservr.exe* in Windows) and the translator (called *otrans* in UNIX and *otrans.exe* in Windows).

   UNIX Users

   To access these dates, type at the command line:

   ```
   strings <program name> | grep otBuild
   ```

   where <program name> is either "cservr" or "otrans"

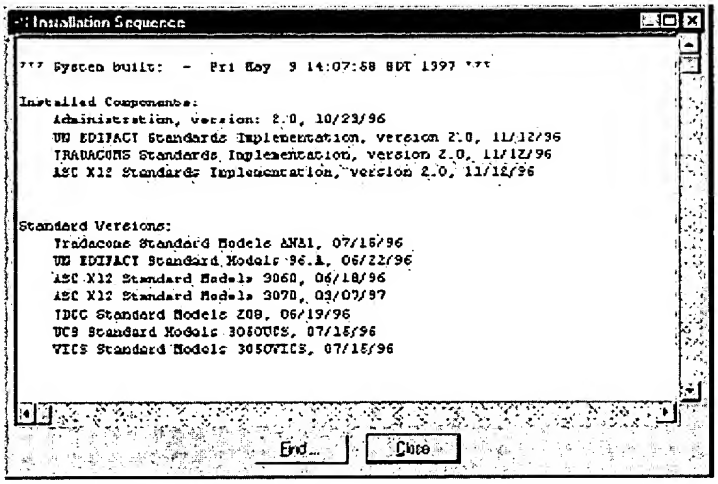
   Windows Users

   To access these dates:

   a. From File Manager or Windows Explorer (Windows 95 or NT 4.0), right-click the filename *cservr.exe* (in Windows).

   b. With the filename highlighted, choose Properties from the File menu. The "Properties" dialog box opens and displays information such as the filename, path, last change, version, copyright, size, and other attributes.

3. Copy down the exact release number of the operating system under which you are running Application Integrator, for example: HP10.01 (include interim release numbers, not simply HP or HP10) or Windows NT 4.0 (include interim release numbers, not just Windows NT).
4. Print the Updates log found by choosing the Updates option on the Help menu of Trade Guide. This log shows the installation sequence, as in the following illustration:



5. Make sure the person placing the support call has taken Application Integrator training and has a thorough background on the issue for which you are seeking assistance.
6. Call the GE Information Services Application Integrator Help Desk for support at (800) EDI-CALL ext. 3005.

Sending a Copy of Your Files to Customer Support

At times, it may be necessary for an example of your problem to be sent to the support staff. For consistency and compatibility across the various platforms that execute Application Integrator, files should be sent to Customer Support as follows:

Backing Up and Restoring UNIX Files

UNIX users should back up models using the UNIX "tar" command.

Try to back up relative, not explicit, so that Customer Support can restore the files into any directory.

Media	Load Command

CD ROM	`mount <devname> <mountpoint>`
	`cp <mountpoint>/<OTsys>/* .`
	`unmount <mountpoint>` (after copy is complete)

tape (either 150 Mb ¼-inch tape, 4 mm DAT tape, or 8 mm DAT tape)	`tar -xvf <devname>`

where:

<mountpoint>	specifies the directory of your system where the device will be mounted to, e.g., "/CDROM" **Note: Sun operating systems do not require a mount to be performed. Sun automatically performs an automount.**
<devname>	specifies the device name of the media drive of your system, e.g., "dev/dsk/c0t5d0" or "/dev/rmt0"
<OTsys>	specifies the name of the directory which stores the files on the CD ROM, either TRANDEV for development systems or TRANPROD for a production/test system.

To restore the files sent from Customer Support, use the tar command with the -xvf options, for example,

```
tar -xvf /dev/fd0
```

Backing Up and Restoring Windows Files

Microsoft Windows has various backup programs available depending on the particular version of the software you are running. Users should either compress all the programs together using programs such as winzip.exe, or send the uncompressed files (assuming they are small in number and size). Refer to your Windows documentation for more information. Be sure to inform Customer Support of the backup/compression program used to send the files and supply them the uncompression program, when necessary.

Customer Support will return the files in a similar format.

Listing the Contents of the Disk or Tape

UNIX Users

To list the contents of the disk/tape, use the tar command with the -tvf options, for example,

```
tar -tvf /dev/fd0
```

Windows Users

There is no method for reviewing the contents of the Windows installation CDs. To review the contents of the installation media, you must first install the product and then use File Manager, Windows Explorer, or the MS-DOS dir command to list the contents of the \Trandev, \Trantest, or \Tranprod directories.

Section 1
XML Overview

Chapter 1 contains information about to how XML relates to Application Integrator, XML Plug-In installation procedures, and background information about inbound and outbound XML processing. To use the XML Plug-In, you should have some knowledge of the XML language and of Application Integrator.

Note: The XML Plug-In is compatible with Application Integrator Version 3.1. If you are using an earlier version, you must upgrade before installing the XML Plug-In.

What is XML?

XML is the 'Extensible Markup Language' - extensible because it is not a fixed format like HTML. XML is defined by the W3C (World Wide Web Consortium) and is designed to enable the use of SGML on the World Wide Web.

- SGML is the Standard Generalized Markup Language, which is used as the international standard for defining descriptions of the structure and content of different types of electronic documents.
- HTML is the Hypertext Markup Language, which is a specific application of SGML used in the World Wide Web.

XML allows groups of people or organizations to create their own customized markup languages for exchanging information in their specialty (be it electronics, engineering, mathematics, music, history, mountain climbing, etc.).

XML is an abbreviated version of SGML to make it easier for you to define your work document types and make it easier for programmers to write programs to handle them. XML files can be parsed and validated like SGML files.

It uses a parser to validate the XML data against a document type definition (DTD). The XML Plug-In contains a data model generator, which uses a valid DTD to create a data model structure to which rules can be added. The parser code operates with the Windows 95, 98, and NT 4.0 operating systems and UNIX Solaris, HP-UX, Digital UNIX, and IBM AIX operating systems.

Installing XML Components

This section provides information and procedures for upgrading your Application Integrator software with XML.

Note: The XML Plug-In is compatible with Application Integrator Version 3.1. If you are using an earlier version of Application Integrator, you must upgrade before installing the XML Plug-In.

XML Installation Overview

The XML software would be installed on your computer system after Application Integrator has been installed in the production, test, and/or development functional areas, and the development seats have been set up. The XML installation procedures should be used to load the XML programs/files into any one of these areas.

We recommend installing the XML software in phases. To load the software into the production and test functional areas, first update the test functional area to review and test the XML installation software as Phase 1. Then load the production functional area in Phase 2.

In a like manner, install a development functional area first to an individual development seat (Phase 1) to become familiar with new features before updating the entire development system and all seats (Phase 2). The program, "otxmlload," allows you to install the entire XML software package at one time or in these recommended phases.

Installing XML to UNIX

The following procedure describes the complete installation of the XML Plug-In on the UNIX operating systems. For more information about installing Application Integrator software, refer to the *Application Integrator Installation Guide*

Application Integrator should already be installed on your computer. For more information about installing Application Integrator, refer to the *Application Integrator Installation Guide.*

Note: Before installing any software, it is a good practice to back up your system.

➢ **To install XML**

1. Log in to the system using the production, test, or development identifier, depending upon the directory you are updating. Confirm that the OT_DIR environment variable is set to the correct path. It should point to the Application Integrator production, test, or development functional area.
2. Create the Application Integrator installation directory by typing:

   ```
   mkdir /tmp/ot
   ```

   This directory name is optional. If there is not enough room in this file system, use a different installation directory. If disk space is plentiful, create one installation directory for production/test and another for development.

Note: If Application Integrator was received on CD or tape, the minimum disk space required is printed at the end of the assembly log (enclosed in the plastic case).

3. Change to the Application Integrator installation directory (or your optional directory) by typing:

   ```
   cd /tmp/ot
   ```

4. Load the software, following the appropriate directions below.

If the XML Plug-In was received on CD, all components of the XML Plug-In will be on one CD. These files will need to be unpacked.

If the XML Plug-In was received on tape, all components of production and test will be on one tape and all components of development will be on a second tape. These files will need to be unpacked.

If you need to unpack files: The files loaded into the installation directory when installed from CD or tape will be "packed" and are named "file1.tar," "file2.tar," etc. These files need to be "unpacked" by typing: `./TAPE.SH` .

Note: For some HP and DEC systems, the extension ";1" is added to the filename TAPE.SH during the installation. In these cases, to issue the command, you must type: `TAPE.SH;1`

If the production/test and the development functional areas are to be updated: *Do not load both development and production/test components into the installation directory at the same time.* Completely update one functional area before loading the software for the other. In order to install the test functional area and review the update before updating production, or to examine the update on an individual development seat before updating development and all seats, it is necessary to do one of the following:

- If disk space is plentiful, use two installation directories, one for production/test files and one for development files. Then, after the first phase of the installation is complete, quit the load which leaves the files in the installation directory until ready to install the second phase.
- If disk space is limited, load production/test and install one or all of the test directories. Complete the load, which will delete the files in the installation directory. Then load development and install one development seat only. Complete the load, which will again delete the files in the installation directory. To install the production directories or the development seats, reload the software from the appropriate media and run the update program again.

Load the media into the installation directory per the appropriate media and operating system using the load command below:

Media	Load Command
CD ROM	mount <devname> <mountpoint> cp <mountpoint>/<OTsys>/* . unmount <mountpoint> (after copy is complete)
tape (either 150 Mb ¼-inch tape, 4 mm DAT tape, or 8 mm DAT tape)	tar -xvf <devname>

Where:

<devname>	specifies the device name of the media drive of your system, for example, "dev/dsk/c0t5d0" or "/dev/rmt0"
<mountpoint>	specifies the directory of your system where the device will be mounted to, for example, "/CDROM"
	Note: Sun operating systems do not require a mount to be performed. Sun automatically performs an automount.
<OTsys>	specifies the name of the directory which stores the files on the CD-ROM, either Trandev for development or Tranprod for production/test.

Check to See if the Control Server Is Running

Before installing the XML Plug-In software, make sure that a Control Server for the functional area being updated is not running. To check to see if a Control Server is running, use the following command:

```
otpsfind cservr
```

If the production system is being updated, cservr *pr* and *pp* should not be running.
If the test system is being updated, cservr *ts* and *pt* should not be running.
If the development system is being updated, cservr *01-50* should not be running.

If a Control Server is running, it must be ended by logging in as the user whose OT_QUEUEID is listed in the results of the `otpsfind` command and typing the `otend` command.

Refer to Chapter 2 of the *Trade Guide for System Administration User's Guide* for more information.

5. Invoke the installation program by typing:

```
./otxmlload
```

The following display appears during the program's execution:

```
************************************************
            GE Information Services
             100 Edison Park Drive
            Gaithersburg, MD 20878
           1-800-EDI-CALL, ext. 3005

     Application Integrator XML Installation
     Production, Test, Development & Seat(s)
************************************************
```

```
Select the system to install the XML
programs/files

   1) Production and Test System

   2) Development System

   Q) Quit this program.
```

If the production and test system were selected, the following menu appears:

```
**************************************************
PLEASE SELECT ONE OF THE FOLLOWING OPTIONS:
1) INSTALL Application Integrator XML INTO
<test/production directory> DIRECTORY
C) COMPLETE THE INSTALLATION OF THIS SELECTION
(This will empty the installation directory
<installation directory>)
Q) QUIT THE INSTALLATION AT THIS TIME

**************************************************
**
Type your selection: (1, C or Q)
and press <Enter> to continue:
```

After the files are installed, select either `C` (to remove the files in the installation directory before exiting) or `Q` (to exit leaving the files in the installation directory for later use in updating the production functional area).

If the development system was selected, the following menu appears:

```
************************************************
PLEASE SELECT WHAT YOU WOULD LIKE TO INSTALL:

1) AN INDIVIDUAL DEVELOPMENT SEAT ONLY
(For single-seat testing of this XML installation
before updating DEVELOPMENT DIRECTORY AND ALL
SEATS)

2) DEVELOPMENT <development directory> AND ALL
ASSOCIATED SEATS
(This option will overwrite single-seat testing)

C) COMPLETE THE INSTALLATION OF THIS SELECTION
(This will empty the installation directory
<installation directory>)

Q) QUIT THE INSTALLATION AT THIS TIME

************************************************
Type your selection: (1, 2, C or Q)
and press <Enter> to continue:
```

After the files are installed, select either C (to remove the files in the installation directory before exiting) or Q (to exit leaving the files in the installation directory for later use in updating the development functional area and all associated seats).

For an individual development seat only, the following menu appears:

```
************************************************
   PLEASE KEY IN THE SEAT NAME OF THE Application
Integrator DEVELOPMENT SEAT YOU ARE INSTALLING.
   THIS MUST BE A SUB-DIRECTORY OF <development
directory>
************************************************
**
   Enter the Directory and
   Press <Enter> to continue:
```

When you have selected what to load, the following messages appear:

```
************************************************

UNCOMPRESSING FILES ... PLEASE WAIT!


COPYING FILES ... PLEASE WAIT!
```

If you selected C, the following paragraphs are output:

```
REMOVING ALL FILES FROM THE INSTALLATION
DIRECTORY!
************************************************
INSTALLATION IS COMPLETE!!  PLEASE REMOVE YOUR
Application Integrator MEDIA AND PUT IN A SAFE
PLACE.
************************************************
```

Note: Previous versions of programs and files that are updated are renamed to end with the tilde (~) character. Since some UNIX operating systems do not allow filenames to be greater than 14 characters, previous versions of programs/files that exceed 14 characters will not exist. After the update is tested, the files ending in a tilde (~) in the production, test, and development directories may be deleted to free disk space.

Installing XML to Windows

The following procedure describes the complete installation of Application Integrator XML on the Windows operating system.

Application Integrator version 3.1 should already be installed on your computer. For more information about installing Application Integrator, refer to the *Application Integrator Installation Guide.*

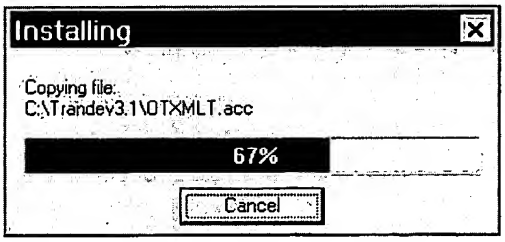
Note: Before installing any software, it is a good practice to back up your system.

➢ **To install XML**

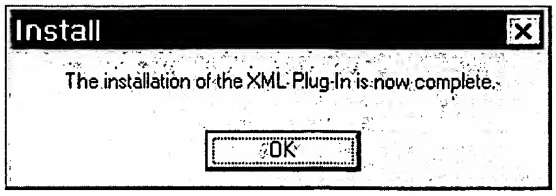
1. Start your computer and run Windows.
2. Insert the CD-ROM and choose Run from the Start menu.
3. From the Run dialog box, type:

   ```
   d:\setup.exe
   ```

   where *d* is the default CD-ROM drive. Choose the OK button.
4. The installation program will detect whether Application Integrator version 3.1 has been installed on your computer.
   a. If the installation program could not detect Application Integrator version 3.1, a warning information box will appear.
   b. If Application Integrator version 3.1 is detected, the installation program will display the following information box while the XML Plug-In software is loading.



5. When the installation is complete, choose the OK button to end the installation.



What are XML Requirements?

XML documents must be *well-formed* and *valid*. Well-formed means the document follows all the notational and structural rules for XML. Programs that intend to process XML should reject any input XML that does not follow the rules for being well-formed. A valid document is valid because it matches its document type definition (DTD).

Well-Formed Documents

There are many rules regarding well-formed documents. Some important rules for creating well-formed documents are as follows:

- **No unclosed tags.** Every start tag must have a corresponding end tag. This is because part of the information in an XML file has to do with how different elements of information relate to one another, and if the structure is ambiguous, so is the information. Therefore, XML does not allow this ambiguous structure.
- **No overlapping tags.** A tag that opens inside another tag must close before the containing tag closes. The structure of the document must be strictly hierarchical. For example: the sequence:

```
<greeting>
Welcome to the <response> world of XML </greeting>
</response>
```

is not well-formed because `<response>` opens inside of `<greeting>` but does not close inside of `</greeting>`. The correct sequence would be:

```
<greeting>
Welcome to the <response> world of XML </response>
</greeting>
```

- **Attribute values must be enclosed in quotation marks.** For example, `<TABLE BORDER=1>` would be valid in HTML but invalid in XML because there are not quotation marks around the attribute value `1`. In XML, a valid attribute value would be `<TABLE BORDER="1">`.

- **The text characters (<), (>), and (") must always be represented by 'character entities.'** To represent these three characters (less than symbol, greater than symbol, and double quotation marks), in the text part of your XML (not in the markup), you must use the special character entities (<), (>), and ("), respectively. These characters are *special characters* for XML. For example, an XML file using a double quotation mark character in the text enclosed in tags in an XML file is not well-formed, and correctly designed XML parsers will produce an error for such input. Additional information about special character coding and processing can be found in Chapter 2, Creating XML Data Models, "Special Characters".

Additional information about well-formed document rules can be found in XML instructional documents and programmers guides.

Valid Documents

While all XML parsers check the well-formedness of documents (meaning the tags are paired and in the proper sequence, attribute values are indicated properly, etc.), some parsers also validate the document. Validating parsers check that the structure and number of tags makes sense.

Case Sensitivity

All of an XML document file, both markup and text, is case sensitive. Element type names, such as those used in start tags and end tags must be defined alike, using either upper- or lowercase characters.

For well-formed files with no document type definition (DTD), the *first occurrence* of an element type name defines the casing. The upper- and lowercase must match; thus, `<IMG/>` and `<img/>` are two different element types.

Attribute names are also case sensitive on a per-element basis. For example, `<PIC width="7in"/>` and `<PIC WIDTH="7in"/>` within the same file exhibit two separate attributes, because the different casings of `width` and `WIDTH` distinguish them.

Document Type Definition Requirements

The document type definition (DTD) is the grammar for a markup language as defined by the designer of the markup language. The DTD specifies what elements may exist, what attributes the elements may have, what elements may or must be found inside other elements, and in what order the elements can appear.

A DTD is associated with an XML document by way of a document type declaration, which appears at the top of the XML file. The document type definition may contain either an internal (inline) copy of the document type definition or a reference to that document as a system filename or URI (universal resource identifier).

DTD Example

The DTD is arranged in hierarchical format. In this example, the hierarchy of the elements is indicated by spaces.

```
<page>
      <head>
         <title>
      <body>
         <title>
      <para>
```

Here, we take this structure and convert it to DTD syntax.

```
<!DOCTYPE page [
      <!ELEMENT page (head, body)>
      <!ELEMENT head (title)>
      <!ELEMENT body (title, para)>
```

Notice how the hierarchy is kept intact within each element.

- `page` is the root element.
- The `page` element consists of a `head` followed by a `body`.
- A `head` element contains a `title` element.
- A `body` element contains a `title` element followed by a `para` element.

This example was coded as it would appear as an internal DTD subset at the start of an XML document. By keeping the DTD inside a document during its development, you can save a lot of file swapping until you are sure the DTD works as intended. You can move the DTD to an external file once it has been finalized.

Data Model Generator Overview

The data model generator is a utility that is used to create the basic structure for a data model. The input to the data model generator would be a well-formed and valid XML file containing a document type definition, either referenced by it or contained within it. From this file, the generator will create data model items for each element identified with formats.

In the following example, the DTD filename is ex10.dtd and is referenced by the XML document. The document contains the minimum requirements: the XML version, the document type, a root start tag (<AddressBook>) and a root end tag (</AddressBook>).

In this example, the document type definition is contained within the XML document. The DTD can be set apart by the <!ELEMENT> and <!ATTLIST> items.

After the data model is generated, the data model items can be manipulated using the Transaction Modeler Workbench application. The generator lays out the appropriate structure for a source or target data model, but the logic within the model must be added using either RuleBuilder or MapBuilder.

The compilation of the generated data model will have a defined structure including element occurrence, field name based on the DTD element, and field length specifications up to the software's maximum limit of 4092. The data model generator creates a general data model structure; with some logic depending upon the nature of the resulting file; additional coding is required to produce a data model that would be valid within a public standard.

Parser

A generic XML parser is a program or class that can read any well-formed, valid XML as its input. The filename of the parser is otxmlcanon.

What is the Parser?

In the product's application, the XML parser takes an incoming stream of XML data from a file or a socket, validates the data and writes the data to the standard output. The output file can then be read and translated by the translator component. The output can be in canonical or non-canonical format. To enable the canonical option, use the -X argument.

Canonical is an industry term that describes the behavior of the parser. When the output is in canonical format, the parser normalizes the XML data by resolving entity references and other markups. It also removes the prolog and DTD references. The document in canonical format is in its lowest form. For example, if you have an entity in your DTD coded:

```
<!ENTITY GPM "XML is easy to use!">
<testcase>&GPM;</testcase>
```

When &GPM; is encountered, the value "XML is easy to use!" would be inserted.

If you ran the parser with the canonical option turned off, the output would be in non-canonical format, and every place you had &GPM;, you would see:

```
<testcase>&GPM;</testcase>
```

If you ran the parser with the canonical option enabled with the -X argument, the output would be in canonical format, and every place you had &GPM; you would see:

```
<testcase>XML is easy to use!</testcase>
```

Notice how the element identifiers are removed. (In this case, the &GPM;.)

XML Special Characters

There are three special characters used within the XML data that cannot be used in data models. These special characters must be identified and changed before the otxmlcanon utility is run. They are the less than symbol (<), the greater than symbol (>), and the ampersand symbol (&). XML uses these three characters, so they cannot appear in your XML data model.

When you must use these characters in your data, such as in a mathematical equation, use the substitution characters in the following table to take the place of these special characters.

XML Special Character	Substitution Character
< (less than symbol)	<
> (greater than symbol)	>
& (ampersand symbol)	&

When these substitution characters appear in a file, they must be recognized and converted into a different symbol so the translator and application can process them as intended.

Shown here is a typical example of how a substitution character should be coded in an XML file element. The program is checking for whether Item_No is less than 5.

```
<ITEM>Item_No &lt; 5</ITEM>
```

Rather than using the less than symbol (<) in the XML code, the substitution character of < was coded.

If the ampersand symbol were needed in an element, it would be coded like this:

```
<DESCRIPTION>Currier&amp;Ives</DESCRIPTION>
```

Rather than using the ampersand symbol (&) in the XML code, the substitution character of & was coded.

Escape and Release Characters

During inbound processing, when the parser sees the substitution character in an XML document, it converts it to the *release character* that was specified on the command line, followed by the intended symbol.

During outbound processing, when the parser sees the *escape character* that was specified on the command line followed by the intended symbol, it converts it to the corresponding substitution character.

otxmlcanon

The otxmlcanon parser is a separate executable that is invoked to read input from a file, socket or standard input and write output to the standard output.

Invoking the Parser

All XML data will be passed to the otxmlcanon parser. If an error is detected by the parser while processing the document, the error will be written to the standard output with a special error tag. Processing of data will terminate.

The parser command has the following syntax:

UNIX operating system

```
otxmlcanon [-r <release character>]
  [-o <escape character>] [-X]
  [<input specification>]
```

Windows operating system

```
otxmlcanon.exe [-r <release character>]
  [-o <escape character>] [-X]
  [<input specification>]
```

Where

Option	Description
-r <release character>	This argument is optional and can be used for both socket and file. It indicates that a release character is present and identifies the release character.
-o <escape character>	This argument is optional and can be used for both socket and file. It indicates that an an output file has been created and identifies the escape character.
-X	This option specifies that the output will be in canonical format.
<input specification >	Indicates the filename of the input file in standard URL format, the socket specification, or if no input specification is entered, the standard input is used. File specification is optional and is in the standard URL format. The full path must be included. For example, file:/C:/appl/rn.xml The socket specification is optional and indicates input will be coming from a socket and identifies the socket. Refer to chapter 4, "Creating Environments," in the *Transaction Modeler's Workbench User's Guide* for information about specifying sockets. If no <input specification> is entered, standard input (stdin) is assumed.

Why Do We Have A Parser?

The parser checks the input for well-formed XML and writes output in the canonical format. The parser also converts characters that XML uses into characters that are recognizable to the translator and the target application.

Input XML Parser

The input data stream is a series of one or more XML documents. Input can come from a file, a socket, or standard input. When multiple documents are processed, each document in the input stream is parsed and the data output until the end-of-stream is reached or an error occurs.

Output XML Parser

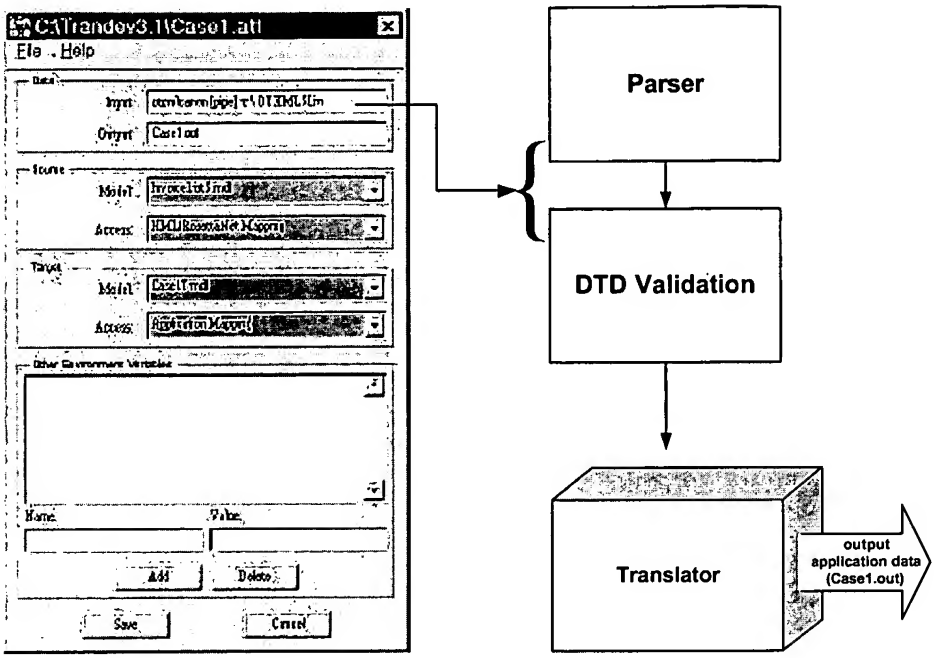
The output from the parser is written to standard output. When multiple documents are parsed, output is in the same order as the input documents.

How XML Works

XML processing depends upon whether the XML document is inbound or outbound.

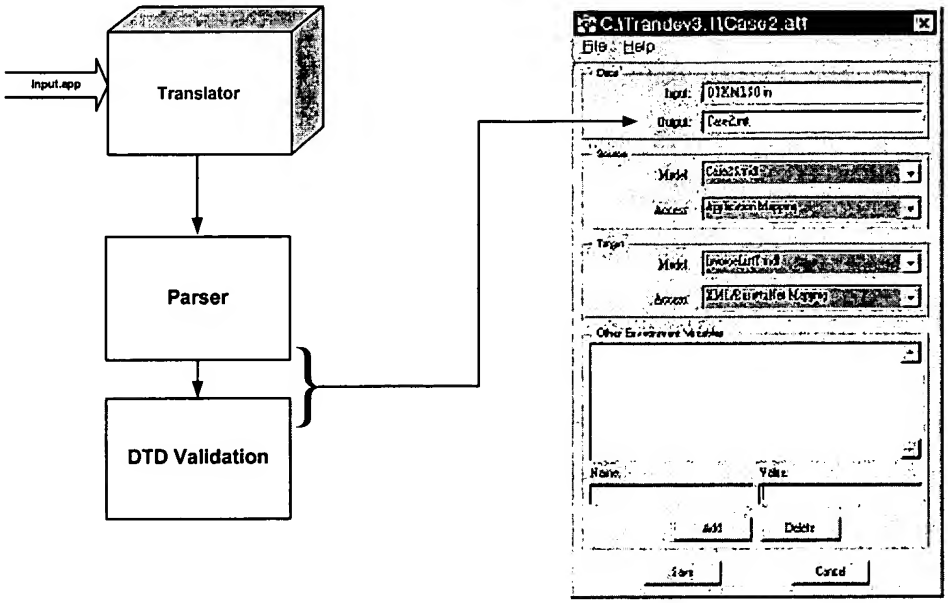
Inbound Processing

During inbound processing, the input file is the XML document that is read by the parser and validated by otxmlcanon. This data is sent to the translator for processing from which an output file of application data is created.



Outbound Processing

During outbound processing, an input application file is sent to the translator for processing. This data then goes to the parser and an XML data file is output.



Section 2
Creating XML Data Models

You can find information about creating generated data models in the "Data Model Generator" section. Information about adding rules to these data models can be found in the "Adding Rules to Generated Data Models" section.

Data Model Generator

Process Overview

The Data Model Generator utility takes a well-formed and valid DTD as input and generates a data model structure. The structure contains the data model item layout, but without rules. The DTD does not have to be local. The XML parser will attempt to load the file from a remote system.

The Data Model Generator will process the root element of a DTD. Since all DTDs have a root element, the generator processes that element only; all other elements in the DTD will be children of the root.

Data Model Generator Requirements

If the document type definition is found on a remote system, or is described using the HTTP format, then the XML parser is relying on the transport protocol of an HTTP server. That is the HTTP server must be installed and operational so the XML parser can retrieve the DTD.

An XML DTD may define a structure that is recursive. Although the data model generator does not *create* a recursive data model structure, the translator supports recursion. The proper data modeling of recursive structure is to place those data model items into a separate data model (either source or target, as appropriate), and use the keyword ATTACH. For additional information about using keywords and advanced data modeling procedures, refer to the *Transaction Modeler Workbench User's Guide.*

Document Type Definition

The document type definition is the starting point for a generated data model. The data model generator will take the DTD and generate a data model. Refer to the "Generated Data Model Examples" section.

Test XML Document Type Definitions

For testing purposes, a document type definition must be developed so it contains each item of the data model structure. The test DTD would be an instance of a fully defined XML document and contain all possible elements, attributes, etc., that appears in the data model.

If the XML is inbound, the data modeler can verify that the entire document has been parsed correctly by the translator.

If the XML is outbound, the output of the translation (that is, an instance of an XML document), may be processed by the parser and validated against the DTD for accuracy.

Creating the Data Model Structure

The data model generator uses the DTD to generate the data model structure. The hierarchy of the data model is controlled by the DTD. If the structure of the generated data model is incorrect, you must either change the DTD and rerun the data model generator, or adjust the data model items using Transaction Modeler Workbench.

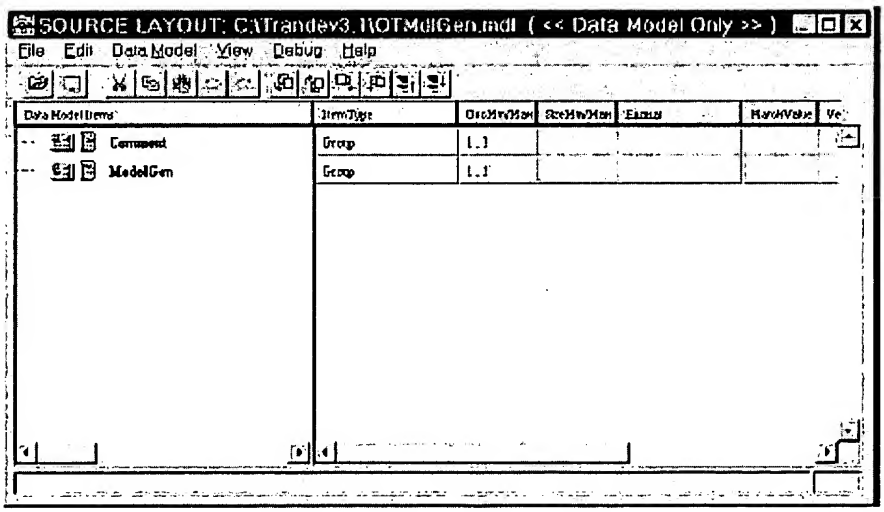
Generating Data Models

Invoking the Data Model Generator

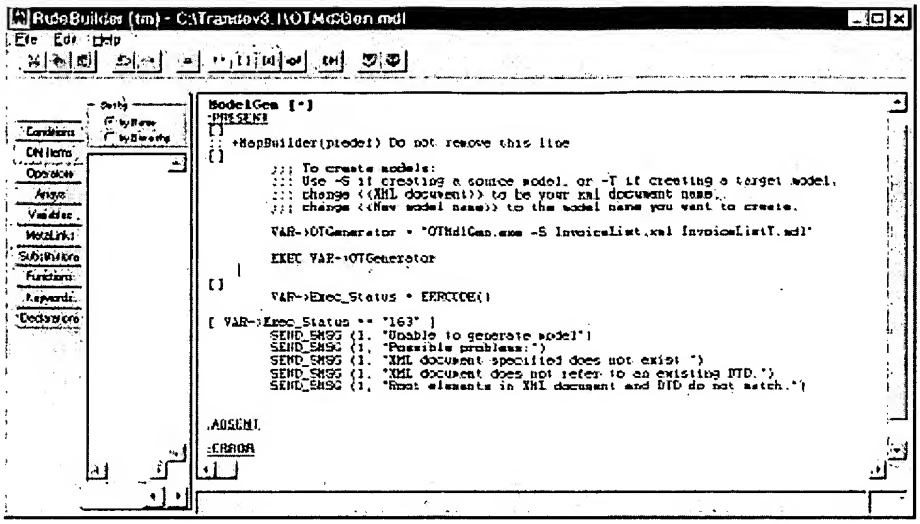
You can invoke the data model generator from the Workbench Run dialog box or from the command line.

➢ **To invoke the data model generator using Workbench**

1. Access the Workbench application according to the appropriate procedure for your operating system.
2. From the File menu, choose Open.
3. Select the map component file OTMdlGen.att. Choose the OK button (UNIX), or the Open button (Windows). The OTMdlGen data model will appear.



4. Choose the ModelGen icon to open RuleBuilder for that data model item. The rules should be displayed.



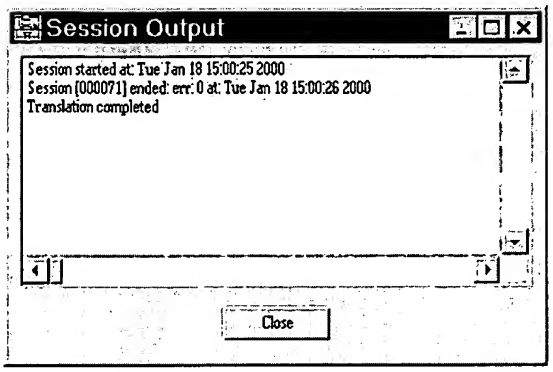
5. Locate the following line of code in the rules:

```
<VAR->OTGenerator="OTMdlGen.exe -S
   InvoiceList.xml InvoiceListT.mdl"
```

6. Type the appropriate argument after the OTMdlGen.exe command to indicate whether the data model to be created will be a source or target model:

Argument	Description
-S	Indicates that the data model will be a source data model.
-T	Indicates that the data model will be a target data model.

7. Apply the new rules, then save the data model.
8. From the Debug menu, choose Run. The Run dialog box will appear.
9. Choose the Save button.
10. Choose the Run button. The translation will execute and the Session Output dialog box will appear. This example shows output from a successful data model generation.



If there are errors, refer to Chapter 3, "Testing and Debugging XML Data Models."

11. Create a map component file to utilize the newly created data model.

➢ **To invoke the data model generator using the command line**

UNIX operating system

```
otmdlgen -S or -T [-I] <input filename>
   <output filename>
```

Windows operating system

```
otmdlgen.exe -S or -T [-I] <input filename>
   <output filename>
```

Where

Argument	Description
-S	Indicates the output should be a source data model.
-T	Indicates the output should be a target data model.
Note: You must use either the –S *or* –T argument to identify the type of output file.	
-I	This argument indicates that the output file should not contain indentations. Required.
<input filename>	Indicates the filename of the well-formed, valid XML document to be used as input to the data model generator. Required.
<output filename>	Indicates the filename of the data model to be generated by otmdlgen. Required

Source Data Model Considerations

The XML document must contain the following items:

- Document type definition. The XML document must either reference the DTD or contain the DTD.
- Start tag of the root element.

XML API Functions

This information is provided for application program interface (API) programmers. The options of otxmlcanon are entered as command line arguments. The XML API has been modified to add the functionality described in the following paragraphs.

SetReleaseChar( )

The SetReleaseChar( ) function takes a character as an argument and stores the value internally. When the release character is set, the PrintBit function modifies the output so the release characters are printed before the special characters.

SetEscapeChar( ) The SetEscapeChar( ) function takes a character as an argument and stores the value internally. When the escape character is set, the parser treats the characters in the #PCDATA following the escape character as regular characters and not as the XML reserved characters. These are output as required by XML.

SkipWhiteSpace( ) The SkipWhiteSpace( ) function allows an application to check for an end-of-stream condition after an XML document is parsed. It can, therefore, be used to restart processing of another XML document after successful processing of an XML document.

Adding Rules to Generated Data Models

After a data model is generated by the Data Model Generator, you will have to add processing rules. These rules would be added using the Transaction Modeler Workbench application. Refer to the *Transaction Modeler Workbench User's Guide* for extensive information about using the application.

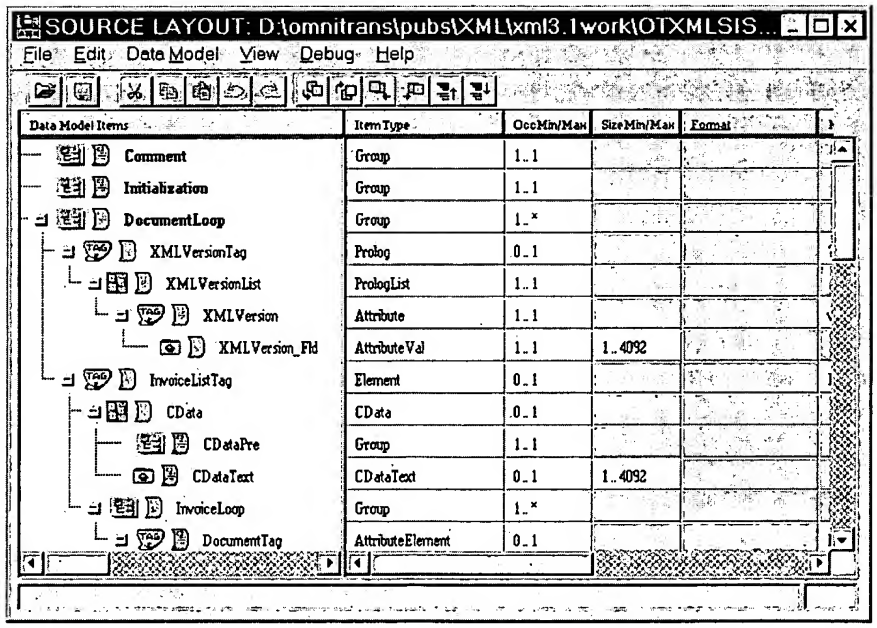
Examples

This section contains two examples of data models: a source layout and a target layout. Each of these examples was generated using the data model generator. All parts of the data model can be edited.

Source Data Model Example

This is a source data model that was created using the data model generator. Each of the data model items was created because they were specified by the DTD. The data model items are associated with the appropriate item type, occurrence, size and formatting/masking characters.

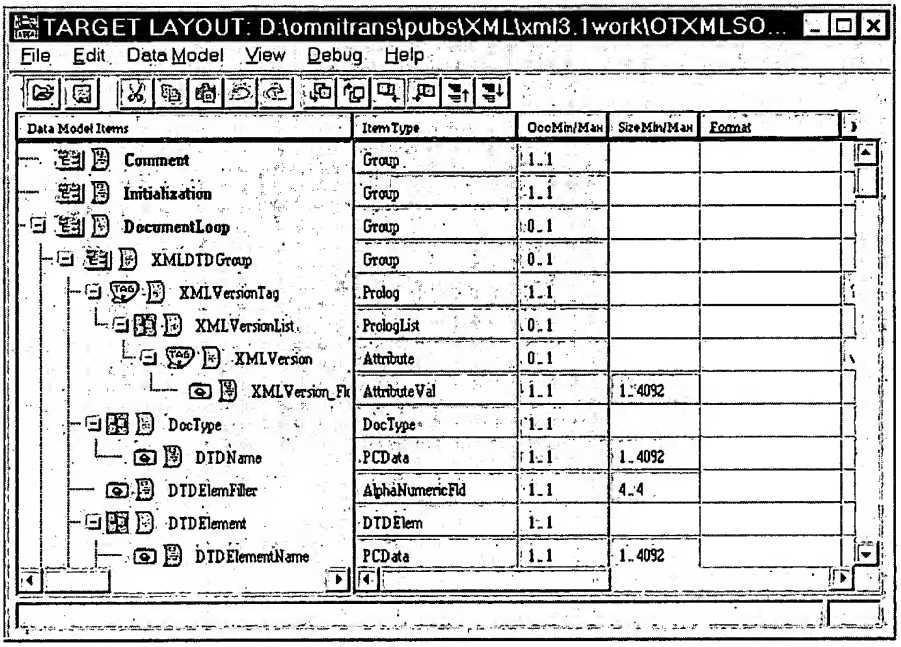
Rules must be added to the data model to accomplish the function of the model. You would add rules using RuleBuilder or MapBuilder.



Target Data Model Example

This is a target data model that was created using the data model generator. Each of the data model items was created because they were specified by the DTD. The data model items are associated with the appropriate item type, occurrence, size and formatting/masking characters.

Rules must be added to the data model to accomplish the function of the model. You would add rules using RuleBuilder or MapBuilder.



Generated Data Model Examples

Shown here are examples of generated data models according to the type of model specified.

Data Model Generator Example 1

In this document type definition example, the one element(FName) becomes 2 data model items; the element field(FNameElement) that carries the tag "FName", and the actual data field(FName_fld) that reads in the #PCDATA data

```
<!ELEMENT AddressBook   (FName, LName) >
<!ELEMENT FName    (#PCDATA)>
<!ELEMENT LName    (#PCDATA)>
<!ELEMENT Address  (#PCDATA)>
<!ELEMENT Phone    (#PCDATA)>
```

This is the data model that would be generated from this document type definition.

```
Initialization {
[]
   SET_SECOND_DELIM(60)
   SET_FIRST_DELIM(62)
}*1 .. 1 ;;|-- end Initialization

AddressBookElement{ Element  "AddressBook"
   FNameElement{ Element  "FName"
      FName_fld{ PCData @1 .. 4092 none
      }*0 .. 1  ;;|-- end FName_fld --|
   }*1 .. 1  ;;|-- end FNameElement --|
   LNameElement{ Element  "LName"
      LName_fld{ PCData @1 .. 4092 none
      }*0 .. 1  ;;|-- end LName_fld --|
   }*1 .. 1  ;;|-- end LNameElement --|
}*1 .. 1  ;;|-- end AddressBookElement --|
```

Note: `<!ELEMENT FName    (#PCDATA)>` **becomes**

```
FNameElement{ Element  "FName"
      FName_fld{ PCData @1 .. 4092 none
      }*0 .. 1  ;;|-- end FName_fld --|
   }*1 .. 1  ;;|-- end FNameElement --|
```

Data Model Generator Example 2

In this document type definition example, the plus (+) character indicates that the FName field occurs from 1 to many times in the data model. This occurrence appears on the FNameElement item not the FName_fld item.

```
<!ELEMENT AddressBook   (FName+, LName) >
<!ELEMENT FName    (#PCDATA)>
<!ELEMENT LName    (#PCDATA)>
<!ELEMENT Address  (#PCDATA)>
<!ELEMENT Phone    (#PCDATA)>
```

This is the data model that would be generated from the example 2 document type definition.

```
Initialization {
[]
   SET_SECOND_DELIM(60)
   SET_FIRST_DELIM(62)
}*1 .. 1 ;;|-- end Initialization

AddressBookElement{ Element  "AddressBook"
   FNameElement{ Element  "FName"
      FName_fld{ PCData @1 .. 4092 none
      }*0 .. 1  ;;|-- end FName_fld --|
   }*1 .. *  ;;|-- end FNameElement --|
   LNameElement{ Element  "LName"
      LName_fld{ PCData @1 .. 4092 none
      }*0 .. 1  ;;|-- end LName_fld --|
   }*1 .. 1  ;;|-- end LNameElement --|
}*1 .. 1  ;;|-- end AddressBookElement --|
```

Data Model Generator Example 3

In this document type definition example, because FName appears in a group, the corresponding data model items (refer to example 1) are now in a grouping item ChoiceGrp. The occurrence of the ChoiceGrp is 1 for 1; the occurrence of the FNameElement is 1 to many. The group happens only once. The item itself can loop.

```
<!ELEMENT AddressBook   ((FName+), LName) >
<!ELEMENT FName    (#PCDATA)>
<!ELEMENT LName    (#PCDATA)>
<!ELEMENT Address  (#PCDATA)>
<!ELEMENT Phone    (#PCDATA)>
```

This is the data model that would be generated from the example 3 document type definition.

```
Initialization {
[]
   SET_SECOND_DELIM(60)
   SET_FIRST_DELIM(62)
}*1 .. 1 ;;|-- end Initialization

AddressBookElement{ Element  "AddressBook"
   ChoiceGrp{
      FNameElement{ Element  "FName"
         FName_fld{ PCData @1 .. 4092 none
         }*0 .. 1  ;;|-- end FName_fld --|
      }*1 .. *  ;;|-- end FNameElement --|
   }*1 .. 1  ;;|-- end ChoiceGrp --|
   LNameElement{ Element  "LName"
      LName_fld{ PCData @1 .. 4092 none
      }*0 .. 1  ;;|-- end LName_fld --|
   }*1 .. 1  ;;|-- end LNameElement --|
}*1 .. 1  ;;|-- end AddressBookElement --|
```

Data Model Generator Example 4

In this document type definition example, the Element FName and Address are in a choice group.

```
<!ELEMENT AddressBook  ( ( FName+ | Address ) ,
LName) >
```

The element FName occurrence becomes 0 to many instead of the 1 to many the plus sign (+) calls for because being in a choice group means that FName, if it occurs, can repeat many times. If FName is not present the element, Address will try to be read. Any repetitions in a choice group will always map with a minimum occurrence of 0.

Remember, the pipe (|)' symbol is a logical OR, so in this example the group `(FName+|Address)` reads FName OR Address. If FName occurred 1 to many times, it would be mandatory which doesn't logically fit with FName OR Address.

```
<!ELEMENT AddressBook  ((FName+|Address), LName) >
<!ELEMENT FName  (#PCDATA)>
<!ELEMENT LName  (#PCDATA)>
<!ELEMENT Address   (#PCDATA)>
<!ELEMENT Phone  (#PCDATA)>
```

This is the data model that would be generated from the example 4 document type definition.

```
Initialization {
[]
   SET_SECOND_DELIM(60)
   SET_FIRST_DELIM(62)
}*1 .. 1 ;;|-- end Initialization

AddressBookElement{ Element  "AddressBook"
   ChoiceGrp{
      FNameElement{ Element  "FName"
         FName_fld{ PCData @1 .. 4092 none
         }*0 .. 1  ;;|-- end FName_fld --|
      }*0 .. *  ;;|-- end FNameElement --|
      AddressElement{ Element  "Address"
         Address_fld{ PCData @1 .. 4092 none
         }*0 .. 1  ;;|-- end Address_fld --|
      }*0 .. 1  ;;|-- end AddressElement --|
   }*1 .. 1  ;;|-- end ChoiceGrp --|
```

```
   LNameElement{ Element  "LName"
      LName_fld{ PCData @1 .. 4092 none
      }*0 .. 1  ;;|-- end LName_fld --|
   }*1 .. 1  ;;|-- end LNameElement --|
}*1 .. 1  ;;|-- end AddressBookElement --|
```

Data Model Generator Example 5

In this document type definition example, the FName elements occurrence is 1 to many (as indicated by the plus symbol (+)), the ChoiceGrp occurrence is 0 to many (as indicated by the asterisk symbol (*)). Because FName is the only element occurring in the ChoiceGrp the DMI occurrence can be 1 to many.

```
<!ELEMENT AddressBook   ((FName+)*, LName) >
<!ELEMENT FName    (#PCDATA)>
<!ELEMENT LName    (#PCDATA)>
<!ELEMENT Address  (#PCDATA)>
<!ELEMENT Phone    (#PCDATA)>
```

This is the data model that would be generated from the example 5 document type definition.

```
Initialization {
[]
   SET_SECOND_DELIM(60)
   SET_FIRST_DELIM(62)
}*1 .. 1 ;;|-- end Initialization

AddressBookElement{ Element  "AddressBook"
   ChoiceGrp{
      FNameElement{ Element  "FName"
         FName_fld{ PCData @1 .. 4092 none
         }*0 .. 1  ;;|-- end FName_fld --|
      }*1 .. *  ;;|-- end FNameElement --|
   }*0 .. *  ;;|-- end ChoiceGrp --|
   LNameElement{ Element  "LName"
      LName_fld{ PCData @1 .. 4092 none
      }*0 .. 1  ;;|-- end LName_fld --|
   }*1 .. 1  ;;|-- end LNameElement --|
}*1 .. 1  ;;|-- end AddressBookElement --|
```

Migrating XML Data Models to Production

Once you have completed the source and target data models, the map component files, and the development testing, you are ready to move your XML electronic commerce application to a test area or production area.

Use the procedures found in Chapter 7, "Migrating to Test and Production Environments," of the *Transaction Modelers Workbench User's Guide.*

Section 3
Testing and Debugging XML Data Models

To test whether a data model that was generated is correct, you must pass an instance of a fully defined XML document. That is, the XML document must contain all possible elements, attributes, etc.

If the XML document is inbound (that is, a source data model), you can verify that the entire document has been parsed correctly by the translator by running a translation. Likewise, if the XML document is outbound (that is, a target data model), then the output of a translation (for example, an instance of an XML document) may be processed by the parser and validated against the DTD for correctness.

XML Troubleshooting

Parser Error Handling

When the parser is successful, a return code of zero is generated. When an error is encountered, a non-zero error code is returned by the otxmlcanon utility. The parser writes the error message to the standard output in the following format.

```
<PARSER_ERROR><error code>:<error message
[;detailed error message]> </PARSER_ERROR>
```

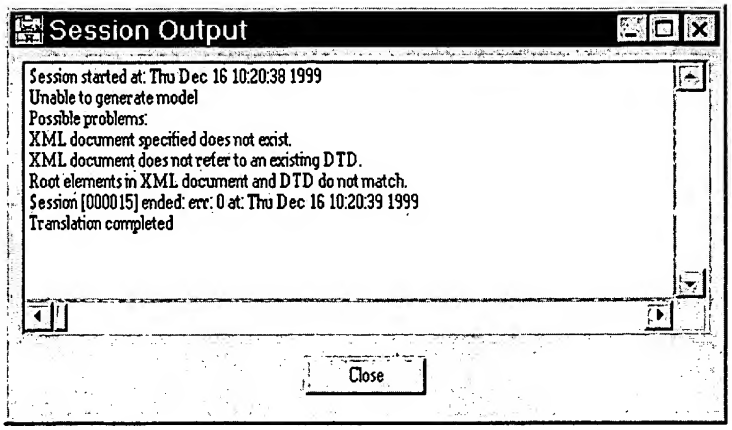
Where

Message Part	Description
<error code>	This is an integer indicating the error code .
<error message[;detailed error message]>	The error message is a short generic description of the error. The detailed error message occurs when the parser has collected additional information about the error.

The error message is written as soon as the error occurs; therefore, the error message may appear at any location in the output stream. The parser exits as soon as an error is detected and the error message is written.

Data Model Generator Error Handling

Shown here is an example of a Session Output dialog box from an unsuccessful data model generation.



This list shows the reasons that the Data Model Generator could fail and the resolutions for the errors.

Cause	Resolution
The specified XML document does not exist in the working directory.	Verify that the filename specified for the XML document is correct and that it resides in the working directory.
The specified XML document does not refer to an existing DTD.	Verify that the filename specified for the DTD is correct and that it resides in the working directory or exists in the directory indicated in the path specified.
Root elements in the XML document and the DTD do not match.	Verify that the root elements specified are correct and match.

The Data Model Generator utility will send back the following return codes.

Return Code	Description
zero (0)	Process was successful.
1	The DTD did not parse successfully.
2	File not found.

Section 4
XML Examples

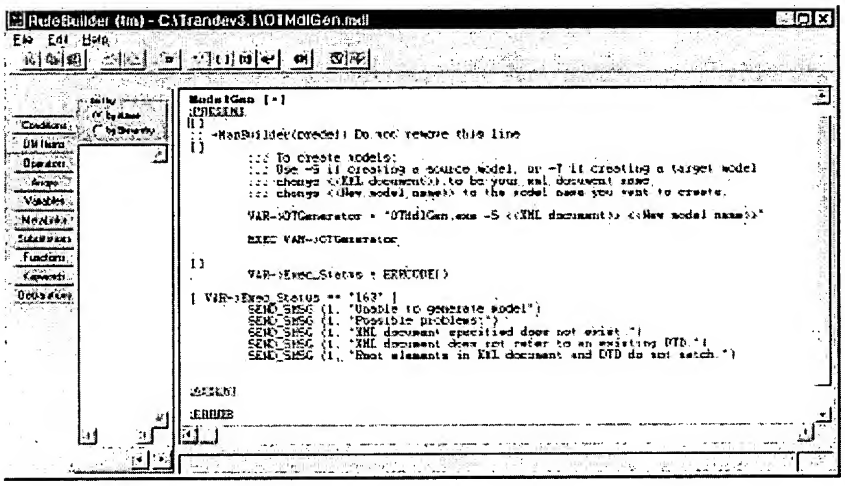
This chapter provides two examples for XML processing. The first example is an inbound example and the second is an outbound example.

Example 1: Inbound Processing

In this example, you will use the data model generator to create an inbound source data model structure. You will also map rules to the data model structure and run a translation. Afterward, you will be able to compare the data models with the DTD (source) and the translation output (target).

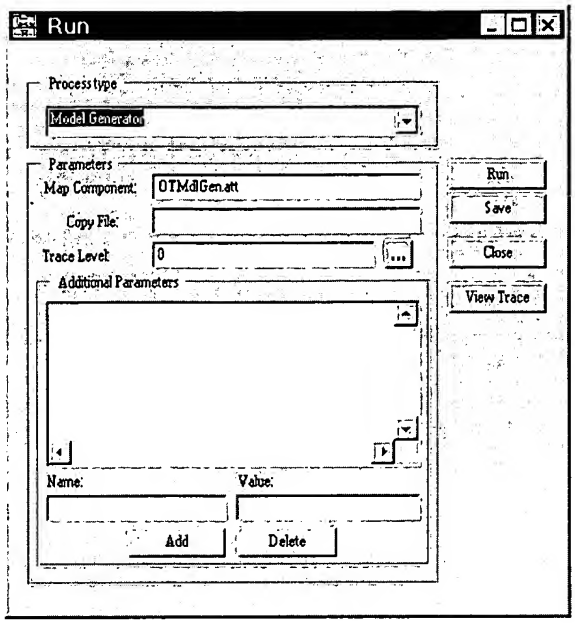
Phase 1: Use Data Model Generator

1. In Workbench, open the OTMdlGen.att map component file.
2. Choose the RuleBuilder icon to open RuleBuilder for the ModelGen data model item.



3. Confirm there is a -S argument after the OTMdlGen.exe command to indicate that the data model to be created should be a source data model.
4. Replace <<XML document>> with InvoiceList.xml.
5. Replace <<New model name>> with InvoiceListS.mdl. The software will create the file when the translation is run.
6. Apply the changes by clicking File then Apply. You can also click the arrow icon on the toolbar.
7. Minimize RuleBuilder.
8. Save the data model.

9. From Layout Editor's menu bar, choose Debug, then Run. The Run dialog box appears.

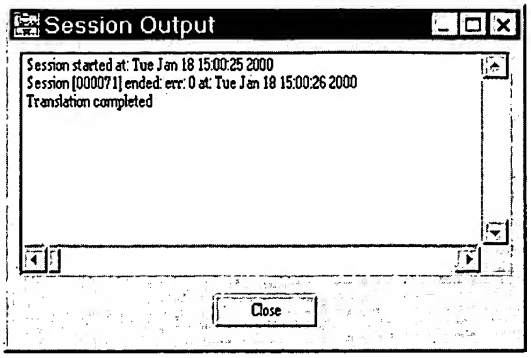


10. Enter the following information to the designated value entry boxes:

Process Type	Model Generator
Map Component	OTMdlGen.att
Trace Level	0

11. Choose the Save button to save the process setup.
12. To run the translation and create the data model, choose the Run button.

13. When the translation completes, a Session Output dialog box appears. It should show no errors.

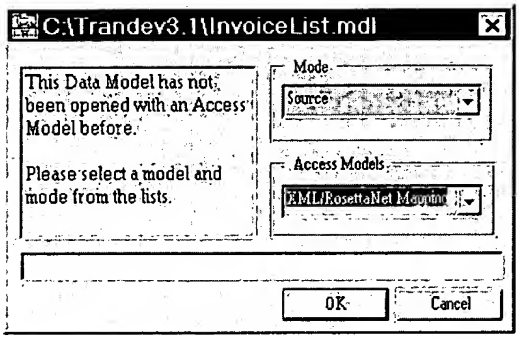


14. Choose the Close button to exit this dialog box.

Phase 2: Data Modeling

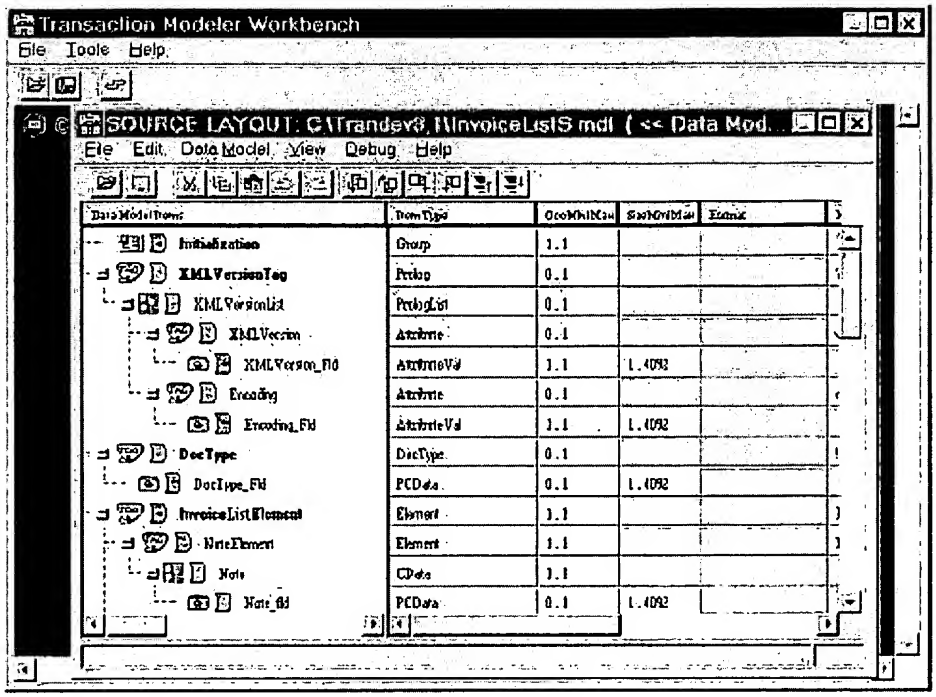
Perform this procedure in the generated InvoiceListS.mdl file.

1. From the Workbench toolbar, choose File, then Open.
2. Highlight the InvoiceListS.mdl filename and choose Open. The Access Model Assignment dialog box will appear.



3. Verify that the Mode value entry box indicates Source.
4. In the Access Models valid entry box, choose XML/RosettaNet Mapping from the drop down menu.

5. Choose the OK button. This opens the InvoiceListS.mdl data model file.



6. Above the Note_fld, add a New Item. Change the New Item's label to Note.
7. Change its Item Type to CDATA.
8. Move the Note_fld right (it should be a child under Note).
9. Save the data model.
10. At the Insert Message Rules/Enveloping message box, choose the Not Used button.
11. In Workbench, open Case1.att. This will open both Case1T.mdl and InvoiceListS.mdl. Arrange the InvoiceListS.mdl and Case1T.mdl screens so they appear side-by-side on the screen.
12. Click the MapBuilder icon on the Workbench toolbar.

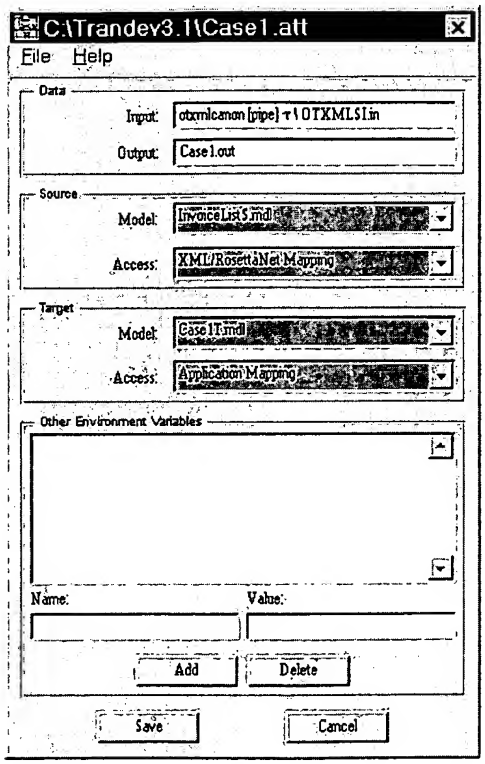
13. Using the following table as a guide, highlight the data model item in InvoiceListS.mdl and move the mouse pointer to the new location in CaseT.mdl and release the mouse button. This copies the rules from the source data model to the target data model. For more information about using MapBuilder, refer to Chapter 3, "Using MapBuilder," in the *Transaction Modeler Workbench User's Guide.*

Note: If you get any loop control message boxes, choose the Close button on the Loop Control information box. (The mouse pointer will look like an hourglass on the message box.)

Source DMI Labels - InvoiceListS.mdl	Target DMI Labels-Case1T.mdl
Note_fld	Comments
Type_fld	DocType
Terms_fld	Terms
IvoiceNumber_fld	InvoiceNumber
InvoiceDate_fld	InvoiceDate
InvoiceTotal_fld	InvoiceTotal
ShipToName_fld	ShipToName
BillToName_fld	BillToName
ItemNumber_fld	ItemNumber (Go To Item)
LineItemsElement	DetailLoop
Quantity_fld	Quantity
Price_fld	Price
Cur_fld	Currency
UnitOfMeasure_fld	UnitOfMeas
PartNumber_fld	PartNumber
Description_fld	Description

14. When all the data model items are mapped, Save the changes to the InvoiceListS.mdl and Case1T.mdl data models.
15. Restore Case1.att. The values in the value entry boxes should appear as shown in the following table:

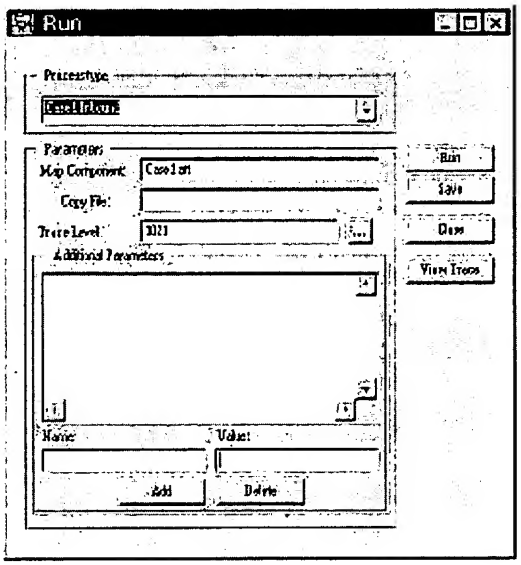
Data Input	otxmLcanon [pipe] -r \ OTXMLSI.in
Data Output	Case1.out
Source Model	InvoiceListS.mdl
Source Access	XML/RosettaNet Mapping
Target Model	Case1T.mdl
Target Access	Application Mapping



16. If you made any changes, choose the Save button. Choose the Cancel button to exit this dialog box.

Phase 3: Run Translation and View Output

1. From the Layout Editor menu bar, click the Debug option.
2. Choose Run. The Run dialog box appears.

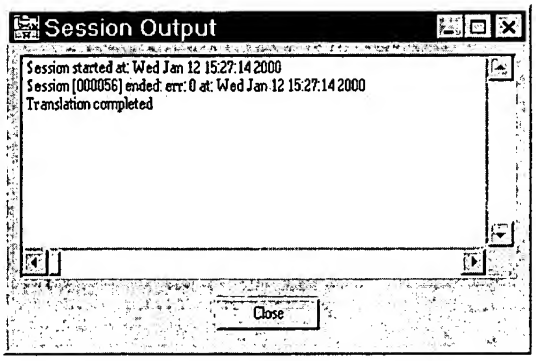


3. Enter the following information to the designated value entry boxes.

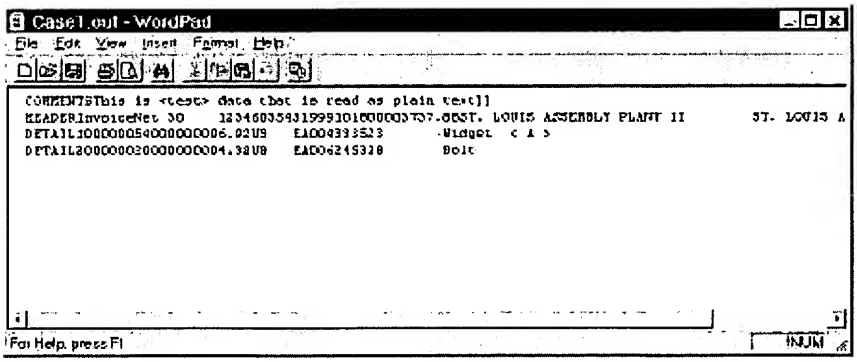
Process Type	Choose **Case1 Inbound** from the drop down list.
Map Component	**Case1.att** This is the map component file to be used in the translation.
Trace Level	**1023** This enables a full trace.

4. Choose the Save button. This saves the process setup.
5. To start the translation, choose the Run button.

6. When the translation is finished, the Session Output dialog box will appear. It should show 0 errors. Make a note of the Session Number.



7. If translation errors occur, debug your data model and rerun the translation.
8. From the View menu of the Layout Editor, select Output File.
9. A text display window appears with the output file for the map component file. Refer to Chapter 6, "Viewing Input and Output Files," of the *Transaction Modeler Workbench User's Guide* for more information.



10. Choose Close to exit the output file.

Example 2: Outbound Processing

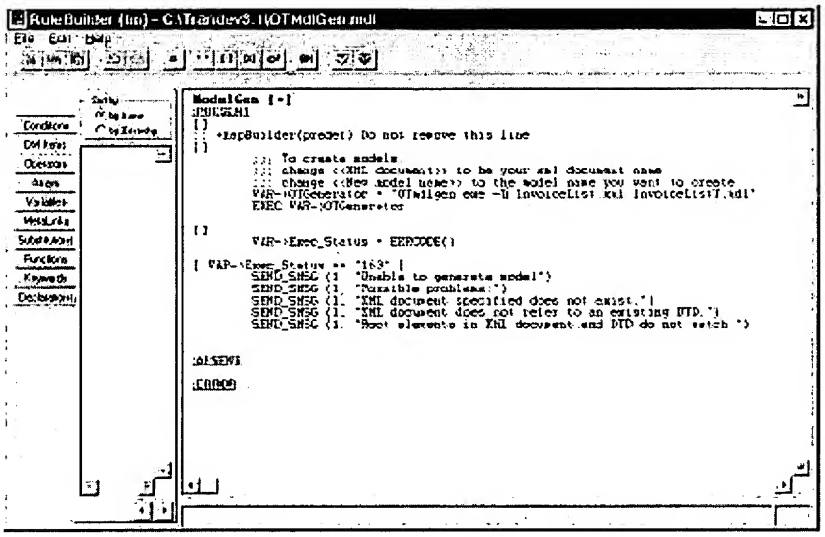
In this example, you will change data model generator so it will process a target data model. You will translate an application file to XML. Afterward, you will be able to compare the output file with the DTD and the translation output.

Note: You must successfully complete Example 1 before performing Example 2.

Phase 1: Use Data Model Generator

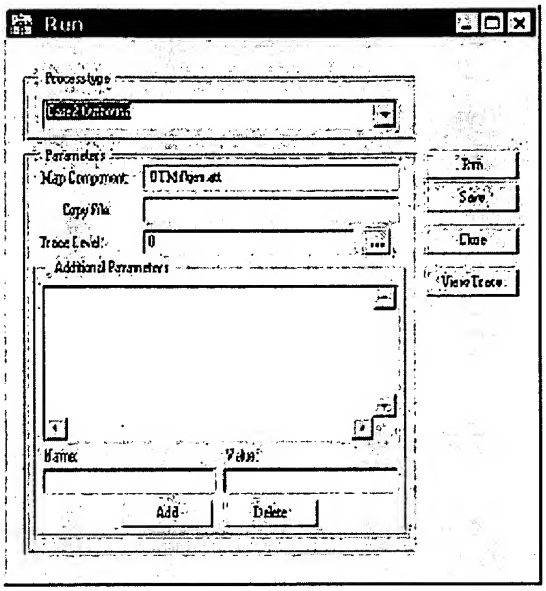
Perform this procedure in the generated InvoiceListT.mdl

1. In Workbench, open the OTMdlGen.att map component file.
2. Click the RuleBuilder icon to open RuleBuilder for the ModelGen data model item.



3. Confirm there is a -T argument after the OTMdlGen.exe command to indicate that the data model to be created should be a target data model.
4. Replace InvoiceListS.mdl with InvoiceListT.mdl. The software will create the file when the translation is run.

5. Apply the changes by clicking File then Apply. You can also click the arrow icon on the toolbar.
6. Minimize RuleBuilder.
7. Save the data model.
8. From the Layout Editor's menu, choose Debug.
9. From the Debug menu, choose Run. The Run dialog box appears.



10. Enter the following information to the designated value entry boxes.

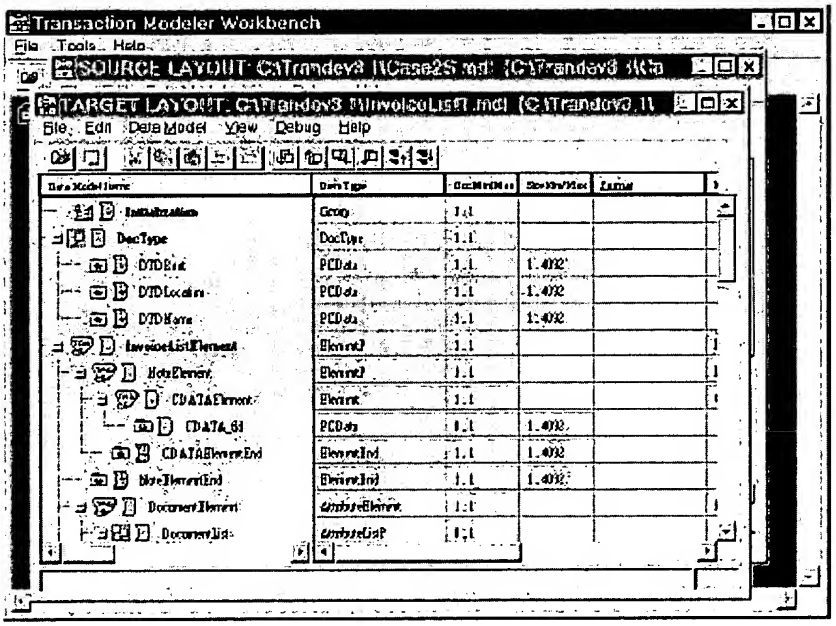
Process Type	Type **Model Generator** or choose from the drop down list.
Map Component	**OTMdlGen.att** This is the map component file to be used in the translation.
Trace Level	**0** This disables the trace.

11. Choose the Save button to save the process setup.
12. To run the translation and create the data model, choose the Run button.

13. When the translation completes, a Session Output dialog box appears. It should show no errors.
14. Choose the Close button to close the Run dialog box.

Phase 2: Data Modeling

1. In Workbench, open Case2.att. This will open Case2S.mdl and InvoiceListT.mdl.



2. In the target data model, above the Note_fld, add a New Item. Change the New Item's label to Note.
3. Change its Item Type to CDATA.
4. Move the Note_fld right (it should be a child under Note).
5. At Size Min/Max, change minimum size on DTDRoot to 12.
6. At Size Min/Max, change minimum size on DTDLocation to 7.
7. Move LineItemsElementEnd right (it should be a sibling to PriceElementEnd).
8. Save the data model.

9. At the Insert Message Rule/Enveloping message box, choose the Not Used button.
10. Click the MapBuilder icon on the toolbar. Arrange the Case2S.mdl and InvoiceListT.mdl windows so they appear side-by-side on the screen.
11. Highlight the data model item in Case1T.mdl and move the mouse pointer to a new location in InvoiceListS.mdl. Release the mouse button. This copies the rules from the target model to the source model.

Note: If you get any loop control message boxes, choose the Close button on the Loop Control information box. (The mouse pointer will look like an hourglass on the message box.)

Source DMI Labels - Case2S.mdl	Target DMI Labels - InvoiceListT.mdl
Comments	Note_fld
DocType	Type_fld
Terms	Terms_fld
InvoiceNo	InvoiceNumber_fld
InvoiceDate	InvoiceDate_fld
InvoiceAmount	InvoiceTotal_fld
ShipToName	ShipToName_fld
BillToName	BillToName_fld
ItemNubmer (Go To Item)	ItemNumber_fld
Detail Loop	LineItemsElement
QtyInvoiced	Quantity_fld
UnitPrice	Price_fld
Currency	Cur_fld
UnitOfMeas	UnitOfMeasure_fld
PartNumber	PartNumber_fld
Description	Description_fld

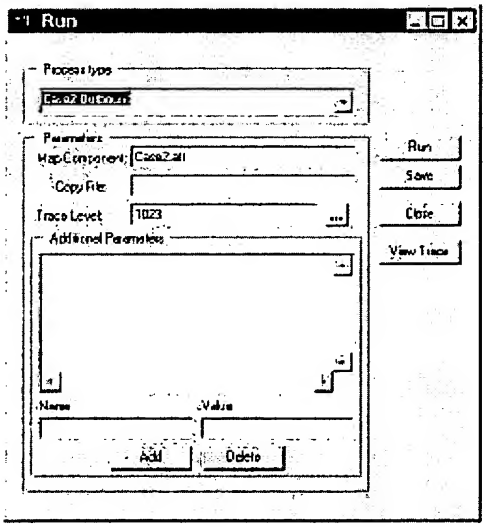
12. When all the data model items are mapped, save the changes to InvoiceListT.mdl and Case2T.mdl.
13. Restore Case2.att. The values in the value entry boxes should appear as shown in the following table:

Data Input	OTXMLSO.in
Data Output	Case2.out
Source Model	Case2S.mdl
Source Access	Application Mapping
Target Model	InvoiceListT.mdl
Target Access	XML/RosettaNet Mapping

14. If you made any changes, choose the Save button. Choose the Cancel button to exit this dialog box.

Phase 3: Run Translation and View Output

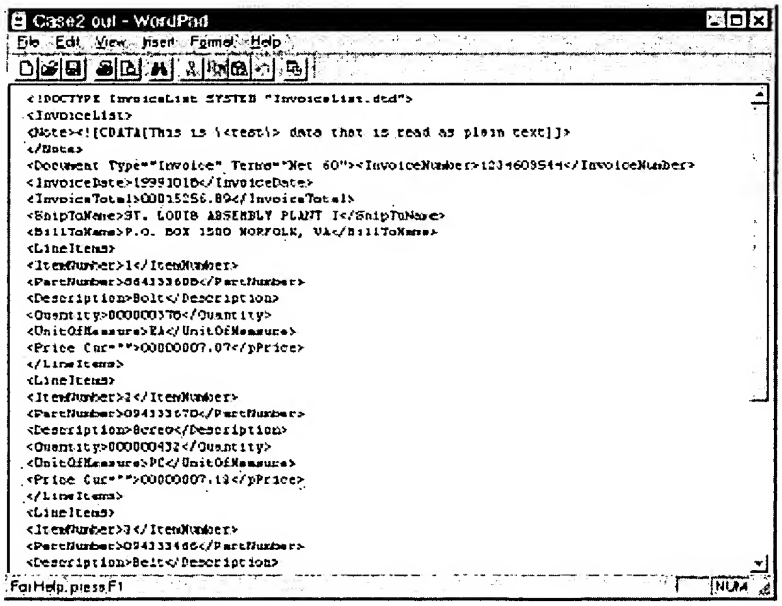
1. From the Layout Editor toolbar, choose the Debug option.
2. Choose the Run button. The Run dialog box appears.



3. Enter the following information to the designated value entry boxes.

Process Type	Choose **Case2 Inbound** from the drop down list.
Map Component	**Case2.att** This is the map component file to be used in the translation.
Trace Level	**1023** This enables a full trace.

4. Choose the Save button. This saves the new process setup.
5. To start the translation, choose the Run button. When the translation completes, you will see the Session Output dialog box. It should show no errors. If translation errors occur, debug your data model and rerun the translation.
6. At the menu bar of the target data model, choose View, then Output File.
7. Choose the output file from the Open dialog box.

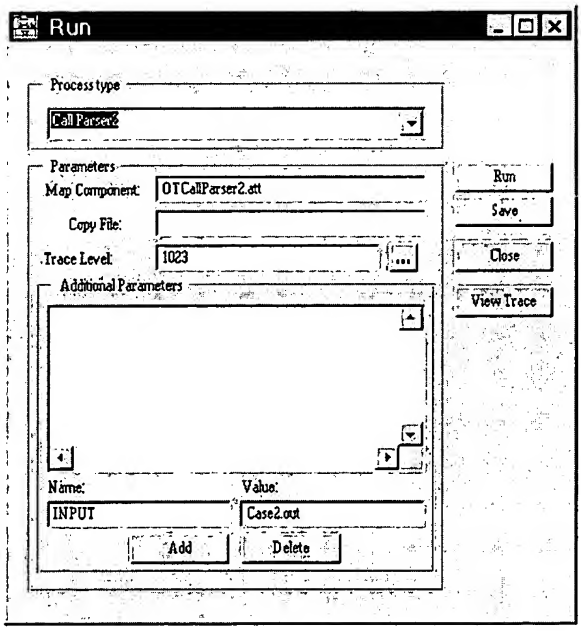


8. Choose Close to exit the output file.

Phase 4: View the Application Data File

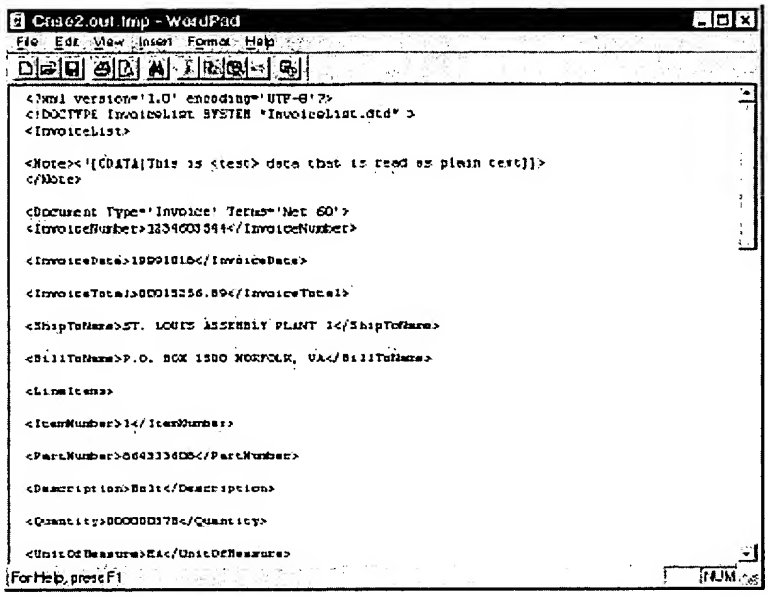
1. Choose Debug, then Run to display the Run dialog box.
2. Enter the following information to the designated value entry boxes.

Process Type	Type **Call Parser2** or choose from the drop down list.
Map Component	**OTCallParser2.att** This is the map component file to be used in the translation.
Trace Level	**1023** This enables a full trace.
Name	**INPUT**
Value	**Case2.out** This specifies the input filename for this translation.



3. Click the Save button to save the process setup.

4. To run the translation, choose the Run button.
5. When the translation completes, you will see a Session Output dialog box. It should show no errors.
6. The output from the parser translation can be found in the Case2.out.tmp file.



Appendix A
XML Plug-In Implementation Files

This appendix provides a list of the files shipped with the XML Implementation.

XML Files Provided

The following list describes map component files (.att), access (.acc), and data model (.mdl) files installed with the RosettaNet Standards Implementation. These files are installed to the working directory and development seat directories.

Filename	Description
Case1.att	Map component file used in Example 1 translation.
Case1T.mdl	Target data model used in Example 1.
Case2.att	Map component file used in Example 2 translation.
Case2S.mdl	Source data model used in Example 2.
InvoiceList.dtd	Document type definition used in examples.
InvoiceList.xml	XML file passed to the data model generator in the examples.
OTMdlGen.att	Creates the environment needed to run OTmdlgen in Workbench.
otmdlgen.exe	Executable which creates a data model structure form the DTD specified in the prolog of an XML document.
OTMdlGen.mdl	Data model needed to run OTmdlgen in Workbench.
otxmlcanon.exe	Windows parser executable used to parse XML data to validate it against the DTD.
OTXMLS.acc	XML source access model used to parse XML data.
OTXMLSI.att	Inbound map component file defining source and target models to parse XML data and create the UDF. It is used by OTXMLI.bat and OTXMLI.sh.
OTXMLSI.bat	Windows batch file used to invoke the sample inbound XML translation.
OTXMLSI.in	Input data for the inbound XML sample (XML file).
OTXMLSI.sh	UNIX shell script used to invoke the sample inbound XML translation.

Filename	Description
OTXMLSIS.mdl	Sample inbound source data model. Used to parse OTXMLI.in.
OTXMLSIT.mdl	Sample inbound target data model used to create UDF output from OTXMLI.bat and OTXMLI.sh.
OTXMLSO.att	Outbound map component file defining source and target models to parse UDF data and create an XML document used by OTXMLO.bat and OTXMLO.sh.
OTXMLSO.bat	Windows batch file used to invoke the outbound sample XML translation.
OTXMLSO.in	Input data for the outbound XML sample (Application file).
OTXMLSO.sh	UNIX shell script used to invoke the outbound sample XML translation.
OTXMLSOS.mdl	Sample outbound source data model used to parse OTXMLO.in.
OTXMLSOT.mdl	Sample outbound target data model used to create XML output from OTXMLO.bat and OTXMLO.sh.
OTXMLT.acc	Target access model used to create XML data.

Appendix B
XML Item Types

The following appendix provides a list and a brief description of item types defined by the Application Integrator access models for RosettaNet and generic models. To reiterate, Application Integrator supplies access models per standards implementation. There are three major access models:

❒ XML/ RosettaNet Mapping	OTXMLS.acc	RosettaNet source access model
❒ XML/RosettaNet Mapping	OTXMLT.acc	RosettaNet target access model
❒ Application Mapping	OTFixed.acc	Generic fixed length data access model

Other access models are installed with each standards implementation.

Note: The "item type" *Group* appears in the Item Type list for all data models. Group is not defined in the access models, since it does not define a data model item used for the parsing or constructing of data. Group allows for the organization of children data model items.

Data Model Item Structures

As discussed in detail in the *Transaction Modeler Workbench User's Guide*, there are four major data model item structures: group, tag, defining, and container items. One or more access model item type names may be associated with each of these structures.


Group

Tag


Defining


Container

XML Access Model Considerations

The access model contains generic definitions for each type of data model item for which data is parsed or constructed. It sets the pre-condition, base condition, and post condition. There are two access models used specifically for RosettaNet data models. The source access model is named OTXMLS.acc and the target access model is named OTXMLT.acc.

XML Source Item Types

The following table lists the items types shipped with the RosettaNet source access model: OTXMLS.acc. You would use these item types when creating a source data model. The Access Model Item Type column indicates the value of the item as it appears in the Item Type drop down menu of the Layout Editor dialog box. The Pre Condition column indicates what the parser will be expecting. The Base Condition column shows the conditions that can be set for the element. The Post Condition column indicates what will follow the element. The Description column gives a short account of each of the elements.

XML Source Access Model Item Types

Access Model Item Type	Pre Condition	Base Condition	Post Condition	Description
AttributeVal	(Quotes)	^(['#' .. '&', '(' .. '~',' ' .. '!']){1 .. *}	(Quotes)	This is used to parse in the double quotes around each attribute value along with the actual value of the attribute.
PCData	(WhiteSpace)	^((#CHARSET)(WhiteSpace))	?	This can be used as a defining to parse in PCDATA in an XML document that may have white space in front of it.
PrologList	(WhiteSpace)	^(CONTAINER)	('?')	This is used to help parse in the prolog which is the XML version line. This container will contain any attributes being sent in the prolog and then parse the question mark (?) at the end of the prolog.
CDataText	?	^(AnyChar)	?	This is used to parse in CDATA. CDATA is a special tag that lets the data contained in this element contain any of the reserved characters in XML.

XML Source Access Model Item Types				
Access Model Item Type	**Pre Condition**	**Base Condition**	**Post Condition**	**Description**
AttributeList	(WhiteSpace)	^(CONTAINER)	(XMLEEndTag || XMLEAEndTag || XMLEATagEnd)	This is used to parse in attributes within an elements beginning tag. The post condition could be any of those specified.
AttributeElement	(%XMLEATag)	^(TAG)	&((XMLEEndTag)	This is used as a parent to parse in an element that contains attributes. The pre condition will parse in the less than sign (<), the element name, and any white space. The post condition will parse in the end tag and the greater than sign (>).
Element	(%XMLEStartTag)	^(TAG)	&(XMLEEndTag)	This is used as a parent to parse in an element that does not contain attributes. The pre condition will parse in the less than sign (<), the element name, and the greater than sign (>). The post condition will parse in the end tag and the greater than sign (>).
Attribute	(%XMLATag)	^(TAG)	(WhiteSpace)	This is used as a parent to parse in the attribute names within an element that contains attributes. The pre condition will parse in the attributes name and the equal sign (=) that follows.
Prolog	(%XMLEATag)	^(TAG)	((XMLProEnd) (WhiteSpace))	This is used as a parent to parse in the prolog. The pre condition will parse in the less than sign (<) and the XML (the match value). The post condition will parse in the greater than sign (>)

XML Source Access Model Item Types				
Access Model Item Type	**Pre Condition**	**Base Condition**	**Post Condition**	**Description**
CData	((White space) (TagStart) ("![CDATA["))	^(CONTAINER)	((TagEnd) . (WhiteSpace))	This is used to parse in a CDATA element. The pre condition will parse in the beginning tag of the CDATA element. The post condition will parse in the the greater than sign (>).
ParserErrFld	?	^(#CHARSET)	?	This is used by inbound processing to parse in <PARSER_ERROR> generated by the parser.
DocType	(%DocTypeTag)	^(TAG)	((TagEnd) (WhiteSpace))	This is used to parse in the DocType Element. The pre condition will parse in the less than sign<) and the match value of !DocType. The post condition will parse in the greater than sign (>).

XML Target Item Types

The following table lists the items types shipped with the RosettaNet target access model: OTXMLT.acc. You would use these item types when creating a target data model. The Access Model Item Type column indicates the value of the item as it appears in the Item Type drop down menu of the Layout Editor dialog box. The Pre Condition column indicates what the translator will write out prior to the data. The Base Condition column shows the conditions that can be set for the element. The Post Condition column indicates what will follow the element. The Description column gives a short account of each of the elements.

XML Target Access Model Item Types

Access Model Item Type	Pre Condition	Base Condition	Post Condition	Description
AttributeVal	(DblQuote)	^([' '.. '!','#'..'~'])[1 .. *]	(DblQuote)	This is used to write out the double quotes around each attribute value, along with the actual value.
PCData	?	^(#CHARSET)	?	This can be used as a defining to write out PCDATA in a RosettaNet document.
PrologList	?	^(CONTAINER)	?	This is used to help write out the prolog which is the XML version line. This container will contain any attributes being written to the prolog.
CDataText	?	^(#CHARSET)	((RtBracket) (RtBracket) (TagEnd))	This is used to write out CDATA. CDATA is a special tag that lets the data contained in this element contain any of the reserved characters in XML. This will also write out the right brackets (]]) at the end of a CDATA element.

XML Target Access Model Item Types				
Access Model Item Type	**Pre Condition**	**Base Condition**	**Post Condition**	**Description**
AttributeElementP	(%XMLEATag)	^(TAG)	&(LineFeed)	This is used when the element itself is a parent. It is used as a parent item to write out an element that contains attributes. The pre condition will write out the less than sign (<), and the element name.
AttributeElement	(%XMLEATag)	^(TAG)	?	This is used as a parent to write out an element that contains attributes. The pre condition will write out the less than sign (<), and the element name.
Element	(%XMLEStartTag)	^(TAG)	?	This is used as a parent to write out an element that does not contain attributes. The pre condition will write out the less than sign (<) and the element name.
ElementP	(%XMLPEStartTag)	^(TAG)	?	This is to be used when the element itself is a parent. It is used as a parent item to write out an element that does not contain attributes. The pre condition will write out the less than sign (<), the element name.
Attribute	(%XMLATag)	^(TAG)	?	This is used as a parent to write out the attribute names within an element that contains attributes. The pre condition will write out the attributes name and the equal sign (=).

XML Target Access Model Item Types				
Access Model Item Type	**Pre Condition**	**Base Condition**	**Post Condition**	**Description**
Prolog	(%XMLEATag)	^(TAG)	((Question) (TagEnd) (LineFeed))	This is used as a parent to write out the prolog. The pre condition will write out the less than sign (<) and the XML (the match value). The post condition will write out the question mark and the greater than sign (?>).
CData	("<!CData[")	^(CONTAINER)	(("]]") (TagEnd) (LineFeed))	This is used to write out a CDATA element. The pre condition will write out the beginning tag of the CDATA element.
DTDAttrTag	(%DTDAttList)	^(TAG)	(LineFeed)	This is used to write out the attributes within the DTD.
DTDElemCon	((Space)(')')	^(PCData)	(')')	This is used to output the contents of an element within the DTD with parenthesis around the contents.
DTDAttList	((TagStart)("!ATT LIST "))	^(CONTAINER)	?	This is used to write out the ATTLIST for the inline DTD.
DTDElem	((TagStart)("!ELE MENT "))	^(CONTAINER)	((TagEnd) (LineFeed))	This is used to write out the DTD elements.
DocType	("<!DOCTYPE ")	^(CONTAINER)	((TagEnd) (LineFeed))	This is used to write out the pre condition of "<!DOCTYPE " for the DOCTYPE element.
AttributeListP	?	^(CONTAINER)	&(TagEnd)	This is used as a parent item for an element which contains attributes. The post condition will write out the greater than sign (>).

Application Integrator ™

XML Schema Plug-In
Version 1.0
User's Guide

October 2001

DISCLAIMER

Information in this document is subject to change without notice. GE Global eXchange Services reserves the right to change (upgrade) data to provide the most accurate, reliable quality product available. Specific mention of a product in this document is not a guarantee by GE Global eXchange Services of complete hardware and software compatibility with your data processing system. If you have questions about hardware and/or software compatibility, please contact your GE Global eXchange Services representative.

The following electronic data interchange (EDI) standards are developed and maintained by these organizations:

TRADACOMS	Article Number Association
X12	Accredited Standards Committee (ASC) and Data Interchange Standards Association (DISA), the secretariat and administrative branch for ASC X12
UN/EDIFACT	Data Interchange Standards Association (DISA), the secretariat and publisher for the Pan American EDIFACT Board (PAEB)

TRADEMARKS

Table of Contents

Preface

The scope of the *XML Schema Plug-In Version 1.0 User's Guide* is to provide data modelers with the information specific to the XML XSD Schema syntax to develop, test, troubleshoot, and move data models that represent XML into a production setting. This manual also provides information about the parser, source and target model examples, and the XSD Schema Data Model Generator.

Any operation dealing with Workbench or testing data models will be found in the Workbench on-line Help system. This preface contains information on:

- XML Schema documentation
- Prerequisites for using XML Schema
- Documentation conventions
- Mouse and keyboard conventions
- Keyboard shortcuts
- On-line Help
- Customer Support

About the XML Schema Plug-In Version 1.0 User's Guide

This guide is designed to give you a working understanding of the operation and capabilities of the software. The information in this document is arranged so you can quickly and easily understand the features, menus, and operations.

The *XML Schema Plug-In Version 1.0 User's Guide* is divided into the following sections and appendixes:

Section	Description
Section 1: XML Schema Overview	Provides an overview of the XML Schema features and terminology, and instructions for installing the product.
Section 2: Creating XML Schema Data Models	Provides information about the data model generator, creating data models, adding rules to generated data models, and generated data model examples.
Section 3: Testing and Debugging XML Schema Data Models	Provides information about debugging parser and translation errors.
Section 4: XML Schema Examples	Provides inbound and outbound examples for XML Schema processing.
Appendix A: XML Schema Plug-In Implementation Files	Lists all the files included with the XML Schema Plug-In.
Appendix B: Unsupported items for XML Schema	Lists the items that are available in XML XSD Schema but are not implemented in the XML Schema Plug-In Version 1.0.

Prerequisites for using XML Schema

System Prerequisites

The following software must be installed and configured before installing the XML Schema Plug-In, Version 1.0 software package.

Version 4.0 of the base product

The XML Plug-In, Version 1.1.3

You must have Internet access to allow the XML parser to find nonlocal XML schema definitions (XSD).

For information on the prerequisite hardware and software necessary to run the product on UNIX® and Windows® operating systems, including Workbench™ and Trade Guide™ components, refer to the *Installation Guide*.

Note: We recommend using the Windows default colors for your Windows applications. Changing or customizing your display colors may result in readability problems when using the product in a Windows environment.

User Prerequisites

It is helpful to have the following background before creating data models that use XML syntax:

- Mouse and graphical user interface (GUI) experience - windows and dialog boxes
- Basic knowledge of your operating system and an on-line editor
- Program concept knowledge, including
  - an understanding of data organization
  - an understanding of data manipulation
  - an understanding of program process flow
  - an understanding of testing and debugging
- Knowledge of electronic data interchange, database management, and systems reporting
  - Knowledge of the XML syntax structure for both document type definitions (DTD) and XML schema definitions (XSD)

Documentation Conventions

Typographical Conventions

Regular	This text style is used in general.
`Courier`	This text style is used for system output and syntax examples.
Italic	This text style is used for book titles, new terms, and emphasis words.
[ ]	Options appearing in syntax descriptions are surrounded by brackets. The brackets themselves should never be typed.

User Input

In this document, anything printed in Courier and boldface type should be entered exactly as written. For example, if you need to enter the term "userid," it will be shown in the documentation as **`userid`**.

Notes, Hints, and Cautions

Notes provide additional information and are boxed inside the text, using the following format.

Note: This is a note.

Hints provide helpful tips on performing operations in a quicker manner. They are formatted in the same way.

Hint: This is a hint.

Cautions provide information on practices or places where you could possibly overwrite data or program files. They appear in the following format.

Caution: This is a caution.

Screen Images

The screen images in this manual were taken from the Windows NT version. If you are running the product on Windows 98, 2000, or UNIX, the actual screens (windows and dialog boxes) may differ slightly in appearance. Differences between operating systems are noted throughout this manual where appropriate.

Tables

Tables appear frequently in this manual and are indicated by headings followed by dark underlining then the body of the table without gridlines (in most cases). The end of the table is indicated by double underlines.

Mouse and Keyboard Conventions

In most cases, you can use either a mouse or a keyboard to enter, view, and manipulate the windows and dialog boxes.

Mouse Usage

If you do not know how to use a mouse, follow the steps below to become more familiar with the mouse and its functions.

1. Hold the mouse with your forefinger on the mouse button (if your mouse has two buttons, use the left button.)

2. Move the mouse back and forth, up and down, and notice how the cursor follows the movement of the mouse. If you run out of room to move your mouse, simply pick the mouse up and set it back down in a position where you will have more room. Moving the mouse without resting it on a tabletop will have no affect on the cursor position.

   You should be familiar with the following terms used to describe mouse actions:

Point:	Position the pointer on a GUI item, for example, on an entry in a list box.
Press:	Type one keystroke or a combination of keystrokes.
Click:	Press and release the mouse button. (This action selects or highlights an item or command.)
Drag:	Hold down the mouse button while moving the mouse to a different position. Then release the mouse button.
Double-click:	Quickly click the mouse button twice over an icon (this action opens an item or initiates an action.) When selecting items, double-clicking on the item achieves the same result as selecting the item and then selecting the associated Apply or Command button.

Release: Let go of the mouse button.

You should also be familiar with the following mouse cursor forms:

↖	Appears when a selection has to be made, for example in a dialog box.
I	Appears when inside of the work area window.
|	Appears when inside the value entry box. This cursor is used to enter alphanumeric information. Dragging this cursor across a field value while holding down the mouse button will highlight the value and allow you to delete it by pressing the Delete key.
⇔	Appears when you reach a window boundary. This arrow can appear pointing up-down, left-right, or diagonally (at corners), depending on the direction the cursor was moving on the screen. This cursor is used to extend the sides of a window.

Keyboard Shortcuts

In many cases, you will have the option of either using the mouse or keyboard to perform an action.

Customer Support

Customer support is offered through a separate Support Services Agreement. The type of support offered is described within the contract. The support services cover:

- Program updates
- Data model updates (for example, generic and public standards-specific data models)
- Standards updates (for example, ASC X12 annual version updates)
- Help Desk Support

Calling for Customer Support

To more effectively help you when you call in for support, follow these steps:

1. If possible, attempt to resolve the question internally or through the product documentation, including the print, on-line, and training documentation.

2. Make sure you have copied down the exact version of the software for which you are seeking assistance. The version is found by selecting the About option on the Help menu. Also copy down the compile version date of the Control Server (called *cservr* in UNIX and *cservr.exe* in Windows) and the translator (called *otrans* in UNIX and *otrans.exe* in Windows).

   UNIX Users

   To access these dates, at the command line, type:

   ```
   strings <program name> | grep otBuild
   ```

   where <program name> is either "cservr" or "otrans"

   Windows Users

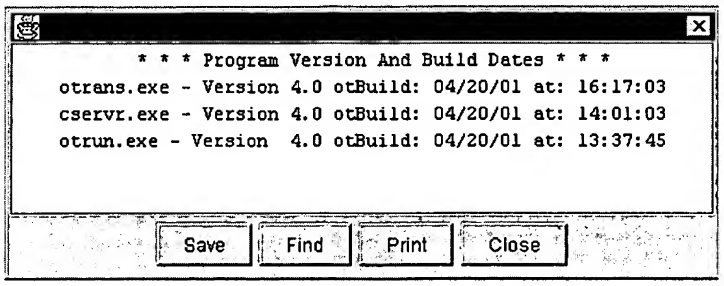
   To access these dates:

   a. From File Manager or Windows Explorer (Windows 98, 2000 or NT 4.0), right-click the filename *cservr.exe*.

   b. With the filename highlighted, choose Properties from the File menu. The "Properties" dialog box opens for CSERVR.EXE and displays information such as the filename, path, last change, version, copyright, size, and other attributes.

3. Copy down the exact release number of the operating system under which you are running the software, for example: HP10.01 (include interim release numbers, not simply HP or HP10) or Windows NT 4.0 (include interim release numbers, not just Windows NT).

4. Print the Updates log found by choosing the Updates option on the Help menu of Trade Guide. This log shows the installation sequence, as in the following illustration:



5. Make sure the person placing the support call has a thorough background on the issue for which you are seeking assistance.

6. Call your software provider.

Sending a Copy of Your Files to Customer Support

At times, it may be necessary for an example of your problem to be sent to the support staff. For consistency and compatibility across the various platforms that execute Trade Guide and Workbench, files should be sent to Customer Support as follows:

Backing Up and Restoring UNIX Files

UNIX users should back up models using the UNIX "tar" command.

Try to back up relative, not explicit, so that Customer Support can restore the files into any directory.

```
mount <devname> <mountpoint>
```

where:

<devname> specifies the device name of the media drive of your system, e.g., "dev/dsk/c0t5d0" or "/dev/rmt0"

<mountpoint> specifies the directory of your system where the device will be mounted to, e.g., "/CDROM"

Note: Sun operating systems do not require a mount to be performed. Sun performs an automount.

To restore the files sent from Customer Support, use the tar command with the -xvf options, for example,

```
tar -xvf /dev/fd0
```

Backing Up and Restoring Windows Files

Microsoft Windows has various backup programs available depending on the particular version of the software you are running.

Users should either compress all the programs together using programs such as winzip.exe, or send the uncompressed files (assuming they are small in number and size). Refer to your Windows documentation for more information. Be sure to inform Customer Support of the backup/compression program used to send the files and supply them the uncompression program, when necessary.

Customer Support will return the files in a similar format.

Listing the Contents of the Disk or Tape

UNIX Users

To list the contents of the disk/tape, use the tar command with the -tvf options, for example,

```
tar -tvf /dev/fd0
```

Windows Users

There is no method for reviewing the contents of the Windows installation CDs. To review the contents of the installation media, you must first install the product and then use File Manager, Windows Explorer, or the MS-DOS dir command to list the contents of the \Trandev, \Trantest, or \Tranprod directories.

Section 1
XML Schema Overview

This section contains information about how XML relates to the software, XML Schema Plug-In Version 1.0 installation procedures, and background information about inbound and outbound XML processing. To use the XML Schema Plug-In Version 1.0, you should have some knowledge of the XML language.

The XML Schema Plug-In development was based on the W3C® specification dated 2 May 2001. It contains the XSD validation and model generation routines.

Caution: Unless specifically instructed in the technical documentation for this product, do not change any system files installed with this product. If a system file is changed, your translations may not run properly.

Modifying system files may result in these types of problems:

Hindered ability to upgrade to future releases.

Obstructed ability to install and run maintenance and patch releases.

Memory handling issues.

Degraded performance (speed) of execution.

Shown here are examples of system filenames:

cbl*	**dummy.***	**sef***
cftest.c	**inittrans.***	**shims***
cservr.*	**uixmrt.***	**tmbench.***
dfapi.*	**otrans.***	**tradegui.***
dfshl.sh	**rms***	**uixm.***

Any filename beginning with "OT" or "ot"

Installing the XML Schema Plug-In

This section provides information and procedures for installing the XML Schema Plug-In.

Note: The XML Schema Plug-In Version 1.0 is compatible with Version 4.0 of the base product. If you are using an earlier version, you must upgrade before installing the XML Schema Plug-In Version 1.0.

Refer to "System Prerequisites" in the Preface section for specific information about items that must be in place before installing the XML Schema Plug-In, Version 1.0.

XML Schema Plug-In Installation Overview

The XML Schema software would be installed on your computer system after the base product has been installed in the production, test, and/or development functional areas, and the development seats have been set up. The XML Plug-In, Version 1.1.3 (or later) must also be installed.

The XML Schema Plug-In installation procedures should be used to load the XML Schema programs/files into any one of these areas. We recommend installing the XML Schema Plug-In in phases. To load the software into the production and test functional areas, first update the test functional area to review and test the XML Schema installation software as Phase 1. Then load the production functional area in Phase 2.

In a like manner, install a development functional area first to an individual development seat (Phase 1) to become familiar with new features before updating the entire development system and all seats (Phase 2). The program aiupdate for UNIX and setup.exe for Windows allows you to install the entire XML Schema Plug-In package in these recommended phases.

Installing the XML Schema

The following procedure describes the complete installation of the XML Schema Plug-In on the UNIX operating systems. For more information about installing the base product software, refer to the

Plug-In to UNIX

Installation Guide.

Note: Before installing any software, it is a good practice to back up your system.

➢ **To install the XML Schema Plug-In**

1. Log in to the system using the production, test, or development identifier depending on the functional area you are updating.
2. Confirm that the OT_DIR environment variable is set to the correct path by typing:

   ```
   echo $OT_DIR
   ```

   It should point to the functional area you are updating.
3. Create the update directory by typing:

   ```
   mkdir /tmp/ot
   ```

   This directory name is optional. If there is not enough room in this file system, use a different installation directory. If disk space is plentiful, create one installation directory for production/test and another for development.
4. Change to the installation directory (or your optional directory) by typing:

   ```
   cd /tmp/ot
   ```
5. Check to see if the Control Server is running by typing:

   ```
   otpsfind cservr
   ```

   If a Control Server is running, it must be ended by logging in as the user whose OT_QUEUEID is listed in the results of the `otpsfind` command and typing:

   ```
   otend
   ```

   Refer to the Trade Guide on-line Help system for more information.
6. Insert the XML Schema Plug-In Version 1.0 CD into the CD-ROM.

7. Load the media into the installation directory by typing:
   mount <devname> <mountpoint>

   where:

<devname>	specifies the device name of the media drive of your system, for example, "dev/dsk/c0t5d0" or "/dev/rmt0".
<mountpoint>	specifies the directory of your system where the device will be mounted to, for example, "/CDROM".

   Note: Sun® operating systems do not require a mount to be performed. Sun performs an automount.

8. The files on the XML Schema Plug-In Version 1.0 CD will be copied to the installation directory by typing:
   . <mountpoint>/unix_install.sh

9. Invoke the update program by typing:
   ./aiupdate

   The following display appears during the program's execution:

```
**************************************************

          Software Update Installation

     Production, Test, Development & Seat(s)

**************************************************
  Select the functional area to be updated:

     1) Production and Test System

     2) Development System

     Q) Quit this program.
```

```
************************************************
```

10. Type your selection The following choices will appear:

```
Please select what you would like to update:

1.An individual seat only (For single-seat testing
   of this update before updating DEVELOPMENT
   DIRECTORY AND ALL SEATS)

2.Development <development directory> and all
   associated seats(This option will overwrite
   single-seat testing)

c)Complete the update of this section
   (This will empty the installation directory
   <installation directory>)

Q)Quit the update at this time

************************************************
```

Type your selection: (1, 2, C or Q) and press <Enter> to continue:

After the files are installed, select either C to remove the files in the installation directory before exiting or Q to exit leaving the files in the installation directory for later use in updating the development functional area and all associated seats.

For an individual development **seat only,** the following menu appears:

```
************************************************
Please key in the seat name of the development
seat you are updating.

This must be a sub-directory of <development
directory>
```

```
******************************************************
```

Enter the Directory and
Press <Enter> to continue:

When you have selected what to load, the following messages appear:

```
******************************************************

Copying files ... Please Wait!
```

If you selected 1 or 2, you will return to the display shown in step 10.

If you selected C, the following paragraphs are output:

```
Removing all files from the installation
directory!

******************************************************

******************************************************

A list of the files updated is contained in
'/tmp/aiupdate.lst'.  Do you want to print the
list now using the 'lp' command?

(the list will remain on disk after exiting this
program)

('tmp/aiupdate.lst' will continue to accumulate a
list of files each time this program is run.
Simply remove or move the file before running this
program to stop the file from growing excessively
large.)

******************************************************

Print this file? (Y or N)?

******************************************************
```

```
Installation is complete.  Please remove your
software media and put it in a safe place.

************************************************
```

Note: Previous versions of programs and files that are updated are renamed to end with the tilde (~) character. Since some UNIX operating systems do not allow filenames to be greater than 14 characters, previous versions of programs/files that exceed 14 characters will not exist. After the update is tested, the files ending in a tilde (~) in the production, test, and development directories may be deleted to free disk space.

Installing the XML Schema Plug-In to Windows

The following procedure describes the complete installation of XML Schema Plug-In to a Windows operating system.

Note: Before installing any software, it is a good practice to back up your system.

Directory Structure

The three functional areas (development, testing and production) are separated from one another through the use of directories. Each functional area is installed in its own directory and subdirectories. By default, the setup program will create three directories:

C:\Trandev	A development directory will include all development programs and the product's processing system (where C is any drive to which you choose to install).
C:\Trantest	A testing directory will include all testing programs. This is used to test programs and

	changes before moving them to production (where C is any drive to which you choose to install).
C:\Tranprod	The main production directory (where C is any drive to which you choose to install).

➢ **To install the XML Schema Plug-In**

This installation displays dialog boxes which prompt you through the installation process.

Note: If the XML Schema Plug-In is already installed on this computer, it must be uninstalled. Refer to "Uninstalling the XML Schema Plug-In" procedure.

1. Start your computer and run Windows. No other software should be running.
2. Insert the XML Schema Plug-In Version 1.0 CD into the CD-ROM.
3. From the Start menu, choose Run.
4. Choose the functional area to be updated.

   To update the Development directory, from the Run dialog box, type:

   ```
   d:\WIN\Development\setup.exe
   ```

   where *d* is the default CD-ROM drive

   To update the Production directory from the Run dialog box, type:

   ```
   d:\WIN\Production\setup.exe
   ```

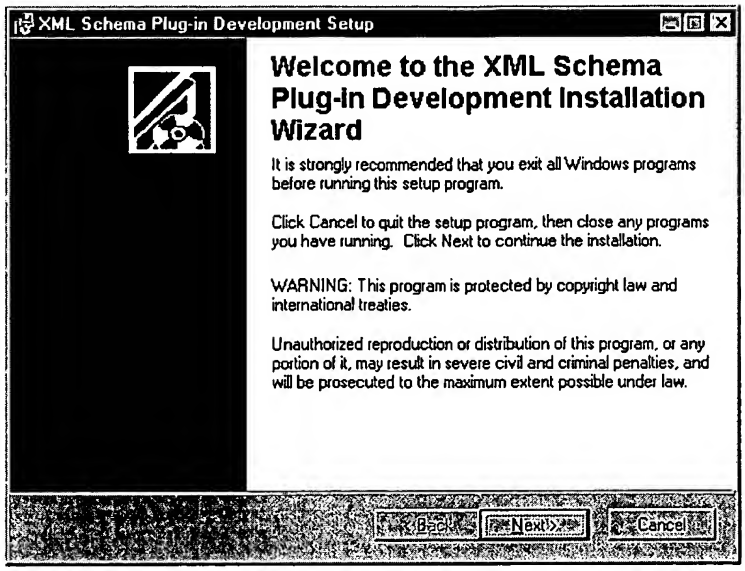
   where *d* is the default CD-ROM drive

   To update the Test directory from the Run dialog box, type:

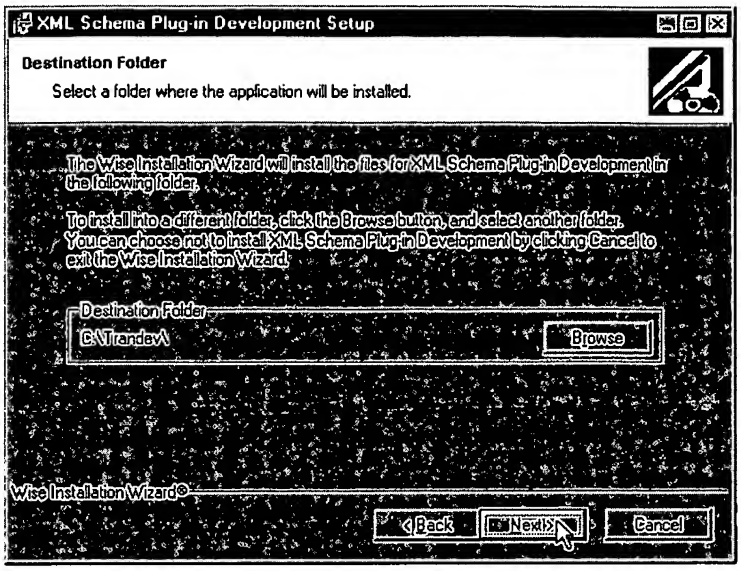
   ```
   d:\WIN\Testing\setup.exe
   ```

where *d* is the default CD-ROM drive

5. Choose the OK button. The following dialog box will appear.



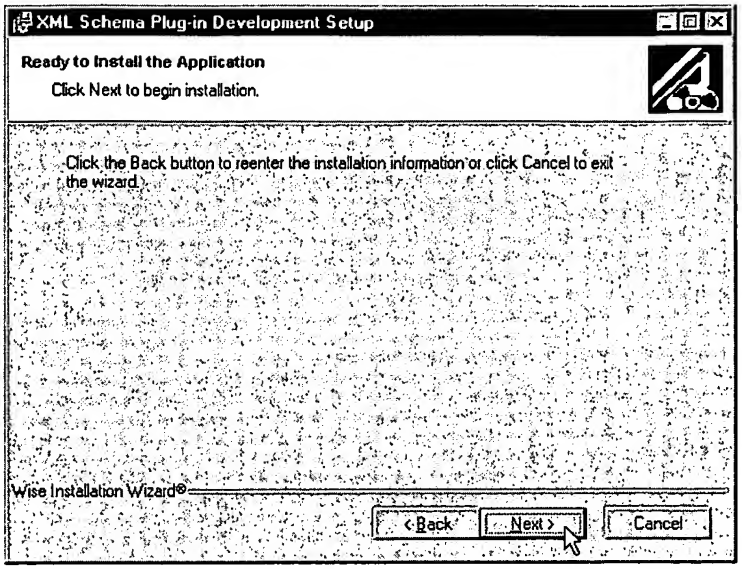
6. Choose the Next button.
7. In the Destination Folder dialog box, review the path designation.



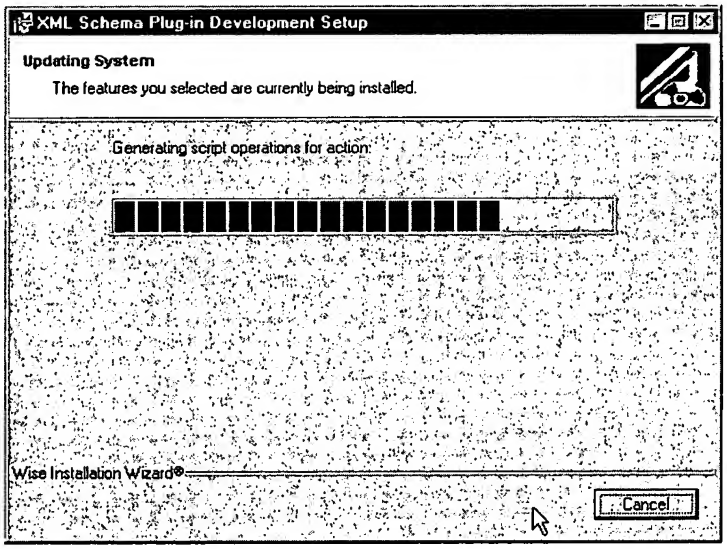
If you want to install the software into this drive and directory, choose the Next button.

If you want to install the software into a different drive and/or directory, choose the Browse button. Locate the drive and directory where the software should be installed. Choose the Open button. The changed drive and directory should appear in the Destination Folder entry.

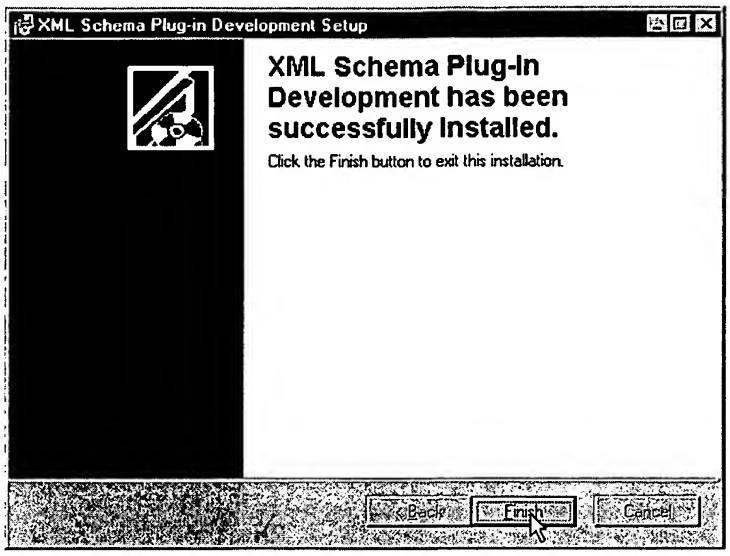
8. Choose the Next button. The Ready to Install the Application dialog box appears. To change any of the settings, choose the Back button. To continue with the installation, choose the Next button.



9. When the Next button is chosen, the Update System dialog box appears and shows the progress of the installation.



10. When the installation is completed successfully, the message "Development has been successfully installed" will appear. Choose the Finish button to exit the installation software.



11. To install the standard into another functional area, repeat steps 3 through 10.
12. Refer to "Configuring the System" to complete the installation procedure.

Uninstalling XML Schema Plug-In Software

If you are reinstalling the XML Schema Plug-In, the previously installed XML Schema Plug-In must be uninstalled (removed) before installing the XML Schema Version 1.0.

➢ **To uninstall the XML Schema Plug-In**

This procedure displays dialog boxes which prompt you through the uninstall process.

1. Start your computer and run Windows. No other software should be running.
2. Insert the XML Schema Plug-In Version 1.0 CD into the CD-ROM.
3. From the Start menu, choose Run.
4. Choose the functional area from which to remove the XML Schema Plug-In software.

To uninstall from the Development directory, from the Run dialog box, type:

```
d:\WIN\Development\setup.exe
```

where *d* is the default CD-ROM drive

To uninstall from the Production directory from the Run dialog box, type:

```
d:\WIN\Production\setup.exe
```

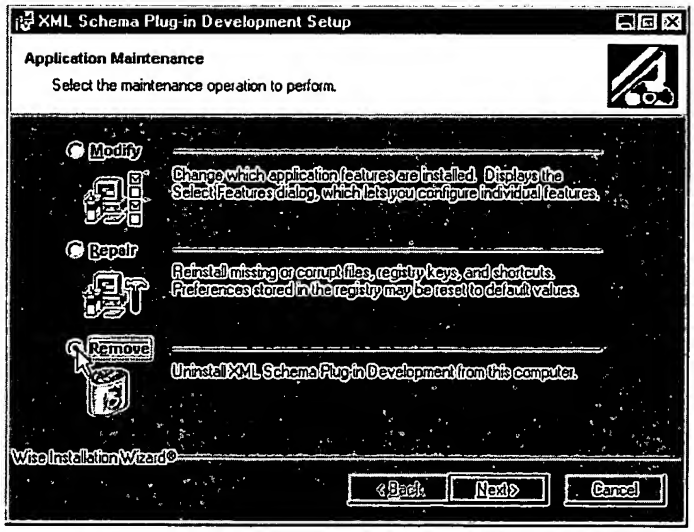
where *d* is the default CD-ROM drive

To uninstall from the Testing directory from the Run dialog box, type:

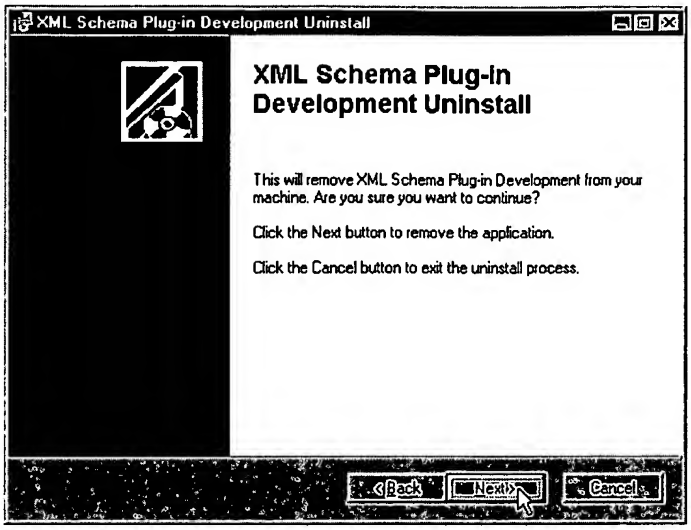
```
d:\WIN\Testing\setup.exe
```

where *d* is the default CD-ROM drive

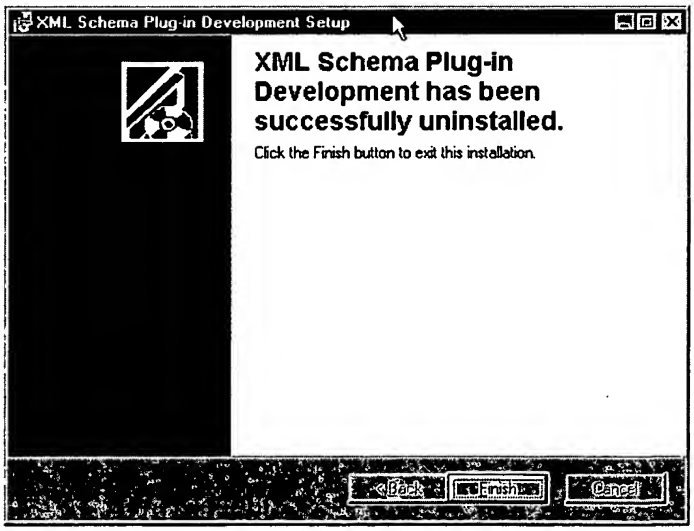
5. Choose the OK button. The Application Maintenance dialog box will appear.
6. Choose the Remove option.



7. Choose the Next button to proceed with the XML Schema Plug-In uninstall.



8. Choose the Next button. The XML Schema Plug-In that was previously installed for the functional area indicated in Step 4 will be removed.



9. Choose the Finish button. Repeat Steps 3 through 9 to remove XML Schema Plug-In software from other functional areas.
10. To install the XML Schema Plug-In Version 1.0, refer to "Installing the XML Schema Plug-In" for installation procedures.

Configuring the System

Before using the XML Schema plug-in, you must make sure the operating system is properly configured.

Set the Environment Variable

The http_proxy environment variable must be set when Internet access will be required when using the XML Schema plug-in.

When the XSD schema to be used for validation will be accessed remotely through the Internet, the URL reference is placed in the XML instance document. If the http_proxy environment variable is not set, the program will be unable to access the XSD schema document and a "file not found" type error will be generated by the translation.

For Windows applications, the http_proxy would be set using the System Properties dialog box.

Variable	http_proxy
Value	http_gw<:port number>

The <port number> is an optional setting which can be used to identify the port number. If the port number is not set, the value will default to :80.

What is XML?

XML is the 'Extensible Markup Language' - extensible because it is not a fixed format like HTML. XML is defined by the World Wide Web Consortium (W3C) and is designed to enable the use of SGML on the World Wide Web.

SGML is the Standard Generalized Markup Language, which is used as the international standard for defining descriptions of the structure and content of different types of electronic documents.

HTML is the Hypertext Markup Language, which is a specific application of SGML used in the World Wide Web.

XML allows groups of people or organizations to create their own customized markup languages for exchanging information in their specialty (be it electronics, engineering, mathematics, music, history, mountain climbing, etc.).

XML is an abbreviated version of SGML to make it easier for you to define your work document types and make it easier for programmers to write programs to handle them. XML files can be parsed and validated like SGML files.

The XML Plug-In contains the XML parser which is shared by both the XML Plug-In and the XML Schema Plug-In. It also contains DTD validation and data model generation routines.

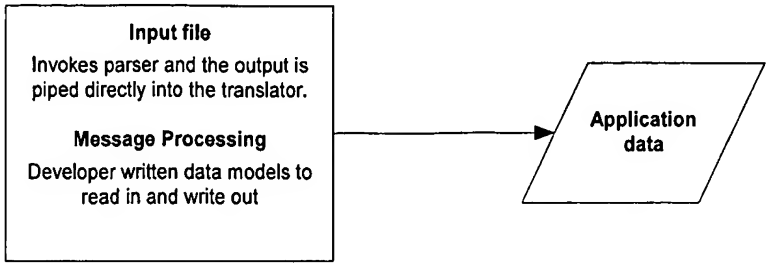
How XML Schema Works

XML Schema processing depends upon whether the XML Schema document is inbound or outbound. There are two methods to process inbound data.

Inbound Processing, Method 1

In the first method, translation occurs with the use of a single environment. The input file is the XML Schema document that is read by the parser and validated by otxmlcanon. This data is sent to the translator for processing from which an output file of application data is created.

The advantage of this method is throughput. The disadvantage of this method is that if an error is encountered, all data processed up to the error will have been translated. Depending on how the translation session is modeled, the data preceding the error may have been updated into other systems or databases. The data would then need to be uncommitted from those systems or databases.

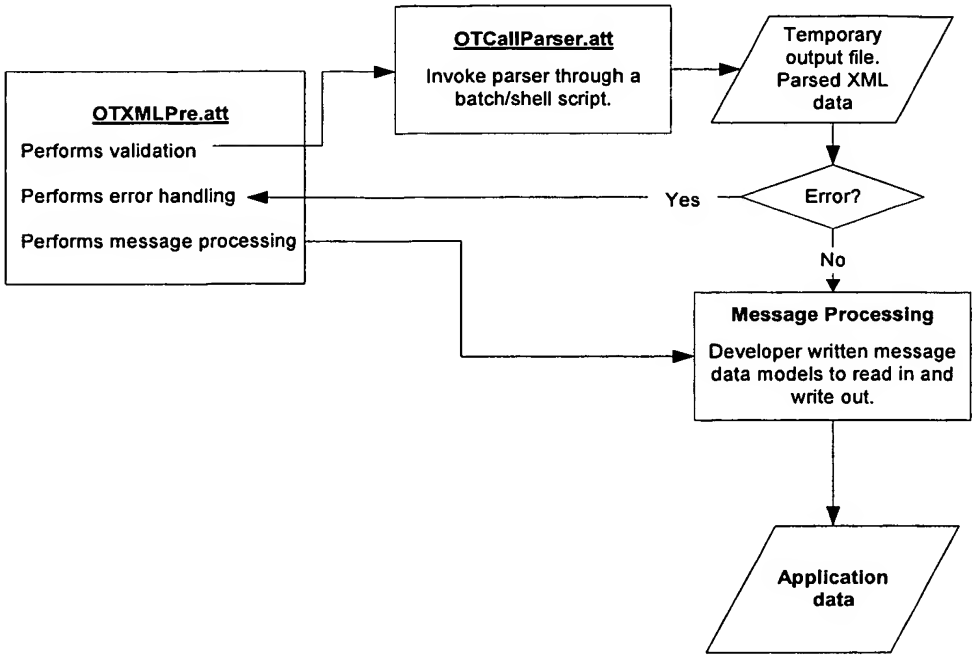


Inbound Processing, Method 2

With the second method of processing, inbound data begins with a pre-translation environment that fully parses and validates the XML data before it is sent through the translator for processing.

The OTXMLPre.att environment attaches to the OTCallParser.att environment to run the otxmlcanon program to check for well-formed XML data and, optionally, validate against an XSD. The output is placed into a temporary file.

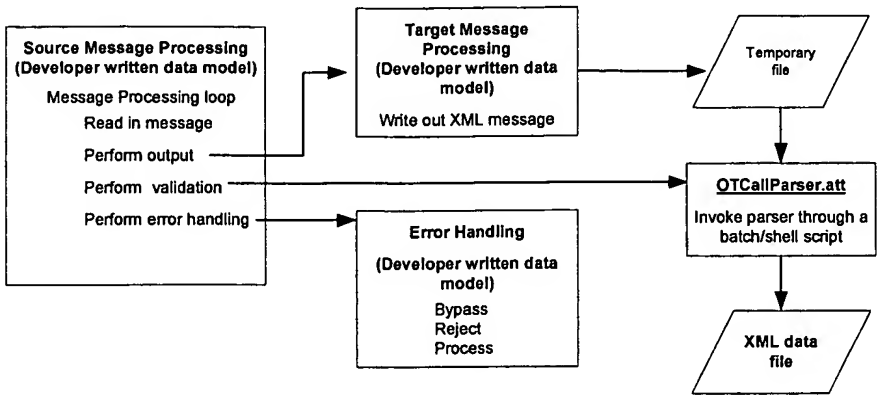
If an error is reported, translation ends or error handling procedures are performed. Upon successful parsing of the data, the OTXMLPre.att attaches to the message processing environment that reads in the XML data and outputs application data. The advantage of this method is that the XML data can be fully checked for well-formedness and, optionally, validated against the XSD before updating any systems or databases. The disadvantage of this method is throughput. Additional time is used to read all of the XML data and write it to a temporary file completely before it is sent through the translator.



Outbound Processing

During outbound processing, an input application file is sent to the translator for processing. Processing begins with a source data model environment, which loops for each message in the input application file until end-of-file is reached.

If no errors are found, the file is sent to the target data model environment. The OTCallParser.att environment is attached to invoke the XML parser. If an error is encountered, error handling procedures are performed. Once the application data has been successfully translated, an XML data file is created.



What are XML Schema Requirements?

XML Schema documents must be *well-formed* and *valid*. Well-formed means the document follows all the notational and structural rules for XML. Programs that intend to process XML schema should reject any XML schema input that does not follow the rules for being well-formed. A valid document is valid because it matches its XML Schema Definition (XSD).

Well-Formed Documents

There are many rules regarding well-formed documents. Some important rules for creating well-formed documents are as follows:

No unclosed tags. Every start tag must have a corresponding end tag. This is because part of the information in an XML file has to do with how different elements of information relate to one another. If the structure is ambiguous, so is the information. Therefore, XML does not allow this ambiguous structure.

No overlapping tags. A tag that opens inside another tag must close before the containing tag closes. The structure of the document must be strictly hierarchical. For example: the sequence:

```
<greeting>

Welcome to the <response> world of XML </greeting>
```

```
</response>
```

is not well-formed because `<response>` opens inside of `<greeting>` but does not close inside of `</greeting>`. The correct sequence would be:

```
<greeting>

Welcome to the <response> world of XML </response>

</greeting>
```

Attribute values must be enclosed in quotation marks. For example, `<TABLE BORDER=1>` would be valid in HTML but invalid in XML because there are not quotation marks around the attribute value `1`. In XML, a valid attribute value would be `<TABLE BORDER="1">`.

The text characters (<), (>), and (") must always be represented by 'character entities.' To represent these three characters (less than symbol, greater than symbol, and double quotation marks), in the text part of your XML data (not in the markup), you must use the special character entities (<), (>), and ("), respectively. These characters are *special characters* for XML. For example, an XML file using a double quotation mark character in the text enclosed in tags in an XML file is not well-formed, and correctly designed XML parsers will produce an error for such input. Additional information about special character coding and processing can be found under the "XML Special Characters" heading in this section.

Additional information about well-formed document rules can be found in XML Schema instructional documents and programmers guides.

Valid Documents

While all XML parsers check that the documents are well-formed (meaning the tags are paired and in the proper sequence, attribute values are indicated properly, etc.), some parsers also validate the document. Validating parsers check that the structure and number of tags makes sense.

Case Sensitivity

All of an XML Schema document file, both markup and text, is case sensitive. Element type names, such as those used in start tags and end tags must be defined alike, using either uppercase or lowercase characters.

For well-formed files with no document type definition or XML schema definition, the *first occurrence* of an element type name defines the casing. The uppercase and lowercase must match; thus, `<IMG/>` and `<img/>` are two different element types.

Attribute names are also case sensitive on a per-element basis. For example, `<PIC width="7in"/>` and `<PIC WIDTH="7in"/>` within the same file exhibit two separate attributes, because the different cases of `width` and `WIDTH` distinguish them.

XML Schema Definition Requirements

The XML Schema Definition (XSD) is the definition of a document in XML syntax. The XSD specifies what elements may exist, what attributes the elements have, what element must be found inside other elements, and in what order the elements can appear.

Namespace Definition

The XSD is associated with an XML document by way of an XML *namespace* definition. A namespace definition is an attribute of the root element. The XML namespace definition may reference a local filename or URI (universal resource identifier).

XSD Example

The XSD is arranged in hierarchical format. In this example, the hierarchy of the elements is indicated by indentations.

```
<page>
  <head>
    <title/>
  </head>
  <body>
    <title/>
    <para/>
  </body>
</page>
```

Here, XML data is converted into an XSD structure:

```
<xsd:schema id="page" targetNamespace="" xmlns=""
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xsd:element name="head">
    <xsd:complexType content="elementOnly">
      <xsd:all>
        <xsd:element name="title" minOccurs="0"
type="xsd:string"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="body">
    <xsd:complexType content="elementOnly">
      <xsd:all>
        <xsd:element name="title" minOccurs="0"
type="xsd:string"/>
        <xsd:element name="para" minOccurs="0"
type="xsd:string"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="page" msdata:IsDataSet="True">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element ref="head"/>
        <xsd:element ref="body"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Notice, through the use of complexType, how the children are defined within their parent:

`page` is the root element.

- The `page` element consists of a `head` element followed by a `body` element.
- A `head` element contains a `title` element.

A `body` element contains a `title` element followed by a `para` element.

Data Model

The data model generator is a utility that is used to create a data model from an XML instance document that points to an XSD. The

Generator Overview

model from an XML instance document that points to an XSD. The input to the data model generator is a well-formed and valid XML file containing a namespace reference for creation from an XSD. From this file, the generator will create data model items for each element identified with formats.

When generating the data model, the .xml file is used as input and requires a specific structure. The following example illustrates an .xml instance document that would be used to generate an .xsd file.

where

purchaseOrder	This is the root element that must be defined globally in the schema file.
xmlns="po1"	Indicates that the XML default namespace is "po1"
xmlns:xsi=	This defines another namespace that will be referred to with the qualifier of xsi.
"http://www.w3.org/2001/XMLSchema-instance"	This identifies the namespace for xsi. This is **not** interpreted as a URL.
xsi:schemaLocation=	This is the namespace qualifier followed by a schema declaration of schemaLocation. Using this declaration, po1, that will be found in the namespace referenced by the xsi qualifier.

po1	This is the default namespace indicated by xmlns="po1".
po1.xsd	This is the URL pointing to the location of the XSD schema file from which the new data model will be generated.

After the data model is generated, the data model items can be manipulated using Workbench. The generator lays out the appropriate structure for a source or target data model, but the logic within the model must be added using RuleBuilder® or MapBuilder™.

Refer to Section 2, "Process Overview," for more detailed information.

The compilation of the generated data model will have a defined structure including element occurrence, field name based on the XSD element, and field length specifications up to the base software's maximum limit of 4092. The data model generator creates a general data model structure; with selected logic depending upon the nature of the resulting file; additional coding is required to produce a data model that would be valid within a public standard.

Parser

A generic XML parser is a program or class that can read any well-formed, valid XML Schema data as its input. The filename of the parser is otxmlcanon.

What is the Parser?

In the product's application, the XML parser takes an incoming stream of XML data from a file or standard input, validates the data and writes the data to the standard output. The output file can then be read and translated by the translator component. The output can be in canonical or non-canonical format.

Additional information about the XML parser can be found in the *XML Plug-In Version 1.1.3 User's Guide*, Section 1, "Parser."

Section 2
Creating XML Schema Data Models

This section contains a processing description for the data model generator, and instructions for creating data models, adding rules to generated data models, and examples of generated data models.

Using the Data Model Generator

Process Overview

The data model generator utility takes a well-formed and valid XSD as input and generates a data model structure. The structure contains the data model item layout, but without rules.

The data model generator determines the root element from an XML instance document, which points to the XSD. The XML document can be very simple containing only the reference to the XSD and the root element. It does not have to be a complete populated document.

Refer to Section 1, "Data Model Generator Overview" for an illustration of an XML instance document.

Data Model Generator Requirements

The XSD does not have to be local. The XML parser will attempt to load the file from a remote system. If the XSD is found on a remote system, or is described using the HTTP format, then the XML parser relies on the transport protocol of an http_proxy server. That is the http_proxy server must be installed and operational so the XML parser can retrieve the XSD.

Document Type Definition

The DTD is a W3C method of defining XML structure.

XML Schema Definition

The XSD expands on the DTD structure. XSD adds extensive data typing capabilities, namespace referencing, and the ability to define this structure in the XML syntax.

Creating the Data Model Structure

The data model generator uses the XSD to generate the data model structure. The XSD controls the hierarchy of the data model. If the structure of the generated data model is incorrect, you must change the XSD, and rerun the data model generator, or adjust the data model items.

Specifying Recursion

An XML XSD may define a structure that is recursive. The data model generator does not *create* a recursive data model structure. The data model is generated to a specified number of recursion levels with rules added to identify when the number of levels is exceeded by a document.

The following rule is inserted into the data model where recursion stops once it reaches the designated <levels>:

```
[ ]
     ;Recursion maximum reached on <DMI label>
     EXIT 357
```

Enumeration and Code Validation

Enumerations define the only acceptable values for elements and attributes in the XSD schema. For example, the element or attribute of "state" might have 50 enumerations for the 50 United States.

The .ids file contains the enumerations. The name of the .ids file is <output filename> with a suffix of ".ids". The records within the .ids file are to updated into the "Standards" directory within Trade Guide, where code lists are placed for validation. These records require unique record keys so that the code lists are associated with a specific document type.

An .ids file is generated when the data model generator is invoked from the command line using the -V, -i <company>, and -j <version> arguments, or using the Workbench Import dialog box.

The company and version are paired and are required only when the XSD contains enumerations and the -V argument is used. These values are used to set the XREF_KEY for the ID code (enumeration) verification and to create records in the generated .ids file. The Company and Version appear on the Trade Guide Standards dialog boxes.

If either the company or the version argument is provided without the other, a message similar to, "No company and/or version supplied for ids file creation. Exiting...". is presented.

Once the data model is generated and the .ids file is created, a message similar to, "Be sure to import the code list file '<output filename>.ids' into Trade Guide using a 'Import Data of Type' of 'Standards'" is presented.

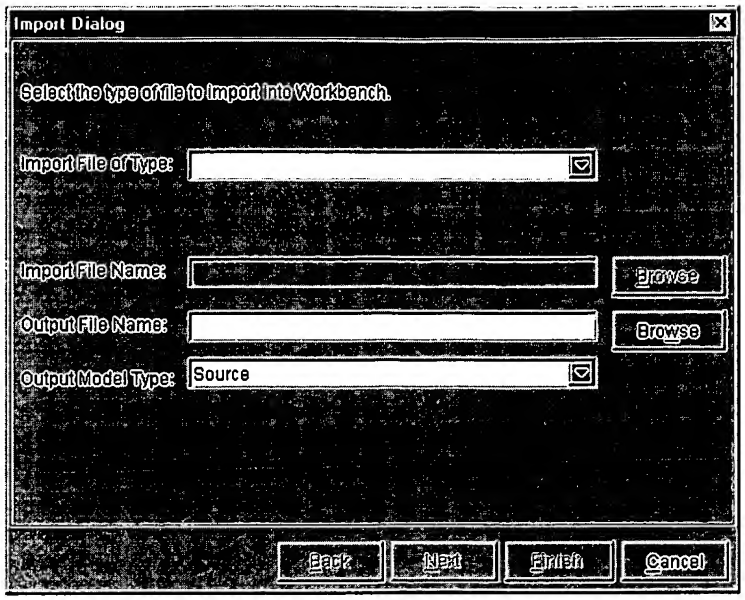
Generating Data Models

The Import dialog box is used to generate data models using the data model generator from the Workbench application. The otmdlgen command is used to invoke it from the command line.

Generating Data Models Using Workbench

➢ **To generate a data model using Workbench**

1. Access the Workbench application according to the appropriate procedure for your operating system.
2. From the File menu, choose Import. The Import dialog box appears.

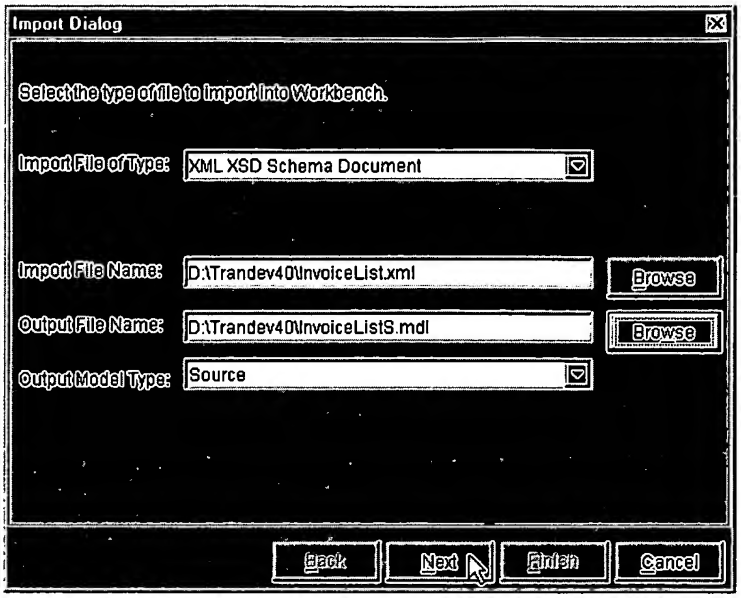


3. At the Import File of Type value entry box, choose XML XSD Schema Document from the list box. This is the instance (.xml) document that contains the reference to the XSD.

4. At the Import File Name value entry box, use the Browse button to locate the document from which the new data model will be created. This should contain the absolute path to the file to be imported.

5. At the Output File Name value entry box, use the Browse button to point to the path for the output file. At the end of the path, type in the filename of the file to be created.

6. At the Output Model Type, choose Source from the list box. Shown here is an example of typical entries to the Import dialog box.



7. Choose the Next button. The 'Enter the following parameters' dialog box appears.

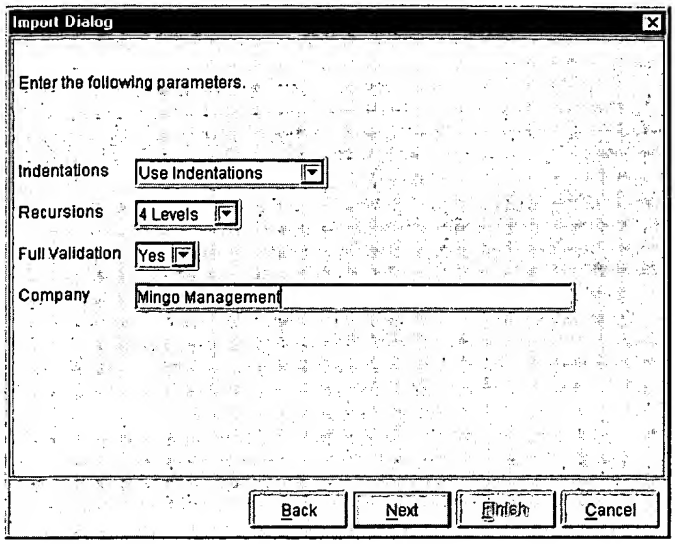
Note: Refer to "Generating Data Models Using the Command Line" for additional information about the value entry parameters.

8. From the Indentations list box, choose either Suppress Indentations or Use Indentations.

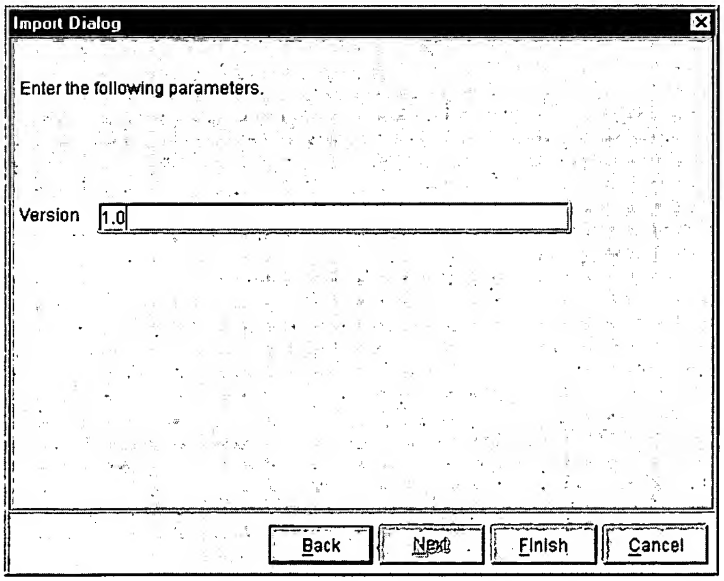
9. From the Recursions list box, choose the number of recursion levels.

10. From the Full Validation list box, choose either Yes or No.

11. At the Company value entry box, type the company name.



12. Choose the Next button. If the XSD from which the data model is to be generated contains enumerations, both the Company and the Version value entry boxes must contain values. A valid version would be 1.0.



13. Choose the Finish button. The Import function will generate a new data model and save it to the location specified in the Output File Name value entry box.

14. The new data model will display in the Workbench work area. Refer to "Source Data Model Example" or "Target Data Model Example" for illustrations of newly generated data models.

Generating Data Models Using the Command Line

➢ **To generate a data model using the command line**

UNIX operating system

From the command line, type:

```
otxsdgen -S or -T [-V] [-I] [-r <levels>] [-i <company>
-j <version>] [-l <locale>] <input_file> <output_file>
```

Windows operating system

From the Run dialog box, type:

```
<path>otxsdgen.exe -S or -T [-V] [-I] [-r <levels>]
[-i <company> -j <version>] [-l <locale>] <input_file>
<output_file>
```

where

Argument	Description
-S	Indicates the output should be a source data model.
-T	Indicates the output should be a target data model.

Note: You must use either the –S *or* –T argument to identify the type of output file.

-V	Indicates that additional validation rules are to be inserted into the data model. Use this argument when the

Argument	Description
	parser is not used in the validation. Otherwise, it is a duplication of some of the parser's validations.
-I	This argument indicates that the output file should not contain indentations. Optional.
-r <levels>	Indicates the number of recursion levels that will be generated in the data model. The default number of levels is 3. Valid number of levels is 1 to 20. Optional.
-i <company>	Type the company name. This company name is used in performing ID code verification and will be placed in an .ids file. This is required when -j <version> is used.
-j <version>	Type 1.0. Identifies the version of the schema document. This version number is used in performing ID code verification and will be placed in an .ids file. This is required when -i <company> is used.
-l <locale>	This argument specifies the locale to be used. Optional.
<input_file>	Indicates the filename of the well-formed, valid XML document to be used as input to the data model generator. Required.
<output_file>	Indicates the filename of the data model to be generated by otxsdgen. Required

The new data model will display in the Workbench work area. Refer to "Source Data Model Example" or "Target Data Model Example" for illustrations of newly generated data models.

Source Data Model Considerations

The XML document must contain the following items:

Reference to an XSD.

Start tag of the root element.

Adding Rules to Generated Data Models

After a data model is generated by the data model generator, you will have to add processing rules. These rules would be added using the Workbench application. Refer to the Workbench on-line Help system for extensive information about using the application.

Rules for Pattern Facets

A pattern facet defined in the schema is not generated into the data model for validation. For a numeric field with a defined pattern, such as a social security number, the data model is generated with a format that does not permit the hyphens to be present. This will cause errors in the data model unless the format is changed.

In the case of the social security number, the defined pattern of "\d{3}-\d{2}-\d{4}" must be changed to "999@-99@-9999" in the data model.

Using Generated Data Models with the Standard Models

In cases where a generated data model is to be used with a standard model, special rules must be placed in the generated data model's PRESENT mode rules.

The following PERFORM statement must be added to the root element's PRESENT mode rules in the source and target data models respectively.

```
PERFORM("OTSrcEnd")

PERFORM("OTTrgEnd")
```

These PERFORM statements assign "Yes" to VAR->OTSourceSuccessful and to VAR->OTTargetSuccessful.

Examples

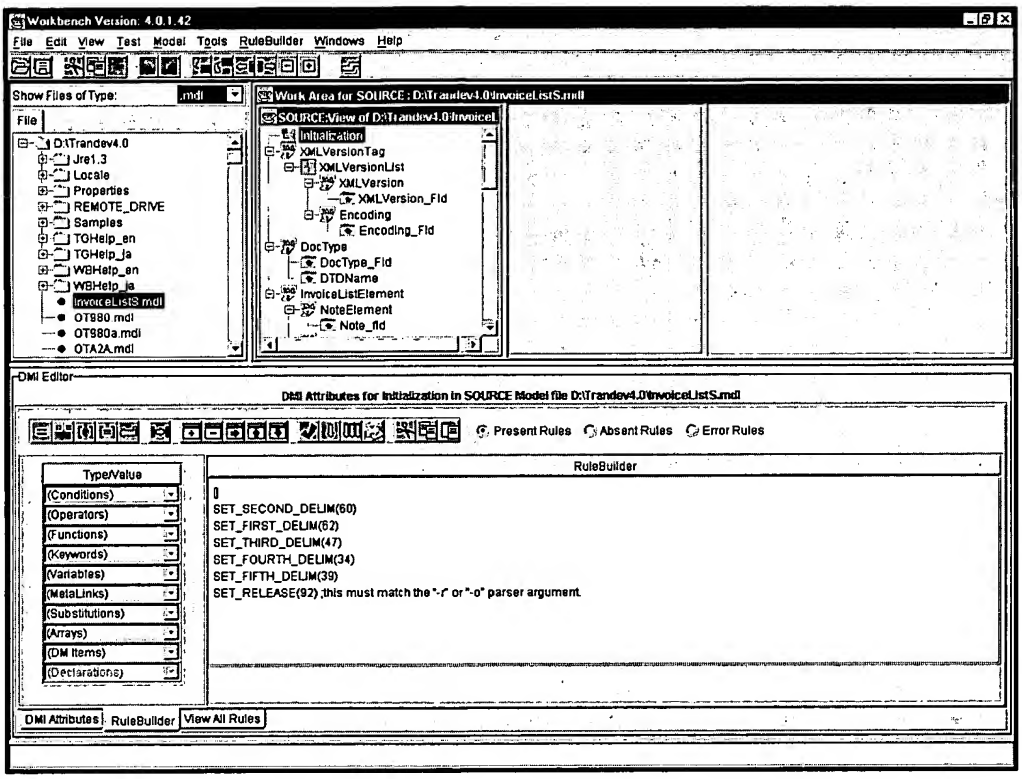
This section contains two examples of data models: a source data model and a target data model. Each of these examples was generated using the data model generator. All parts of the data

model can be edited.

Source Data Model Example

This is a source data model that was created using the data model generator. Each data model item was created because the XSD specified it.

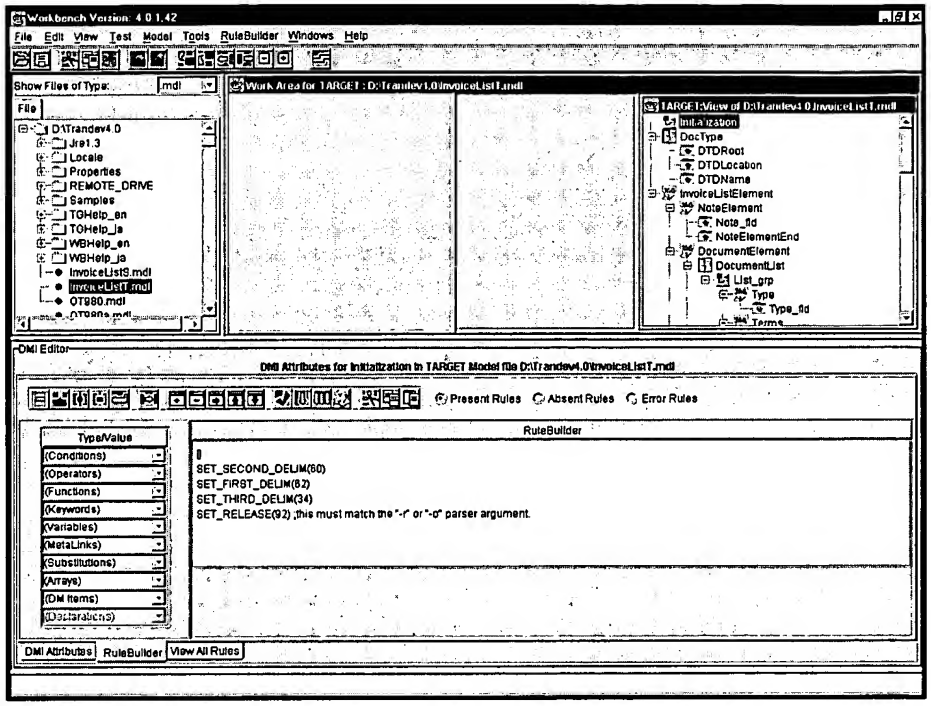
The data model items are associated with the appropriate item type, occurrence, size and formatting/masking characters. Add rules to the data model to accomplish the function of the model using RuleBuilder® or MapBuilder™.



Target Data Model Example

This is a target data model that was created using the data model generator. Each data model item was created because the XSD specified it. The data model items are associated with the appropriate item type, occurrence, size, and formatting/masking characters.

Add rules to the data model to accomplish the function of the model using RuleBuilder® or MapBuilder™.



Generated Data Model Cases

Shown here are data models that are generated according to the type of data model specified.

Data Model Generator Case 1

In this situation, each element becomes two data model items. The FName data model item is generated into FNameElement that carries the tag FName, and the actual data field, FName_fld, that reads in the element's value.

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="po0"
```

```
                targetNamespace="po0"

xmlns:xsi="http:///www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://www.w3.org/2001/XMLSc
hema

http://www.w3.org/2001/XMLSchema.xsd">

 <xsd:element name="AddressBook"
type="AddressBookType"/>

 <xsd:complexType name="AddressBookType">
  <xsd:sequence>
   <xsd:element name="FName" type="xsd:string"/>
   <xsd:element name="LName" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

This is the data model that is generated from Example 1.

```
Initialization {
[]
VAR->OTMdlVer = "StdVer 3.2 - Do Not Remove This
Line"
PERFORM("OTSrcInit",
"STD|XML|(null)~AddressBook~(null)", "XML")
ON_ERROR("xmlOnError")
[]
SET_FIRST_DELIM(62)   ; >
SET_SECOND_DELIM(60)  ; <
SET_THIRD_DELIM(47)   ; /
SET_FOURTH_DELIM(34)  ; "
SET_FIFTH_DELIM(39)   ; '
SET_RELEASE(92)       ; \
}*1 .. 1 ;;|-- end Initialization --|

AddressBookElement { AttributeElement
"AddressBook"
   SeqGroup {
      FNameElement { Element "FName"
         FName_fld { PCData @1 .. 4092 none
         }*1 .. 1 ;; |-- end FName_fld --|
      }*1 .. 1 ;; |-- end FNameElement --|
      LNameElement { Element "LName"
         LName_fld { PCData @1 .. 4092 none
         }*1 .. 1 ;; |-- end LName_fld --|
```

```
        }*1 .. 1 ;; |-- end LNameElement --|
    }*1 .. 1 ;; |-- end SeqGroup --|
}*1 .. 1 ;; |-- end AddressBookElement --|
```

Data Model Generator Case 2

In this XSD case, minOccurs="1" and maxOccurs="unbounded" indicates that the FName element occurs from one (1) to many times in the data model. This occurrence appears on the FnameElement item not the FName_fld item.

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="po0a"
            targetNamespace="po0a"

xmlns:xsi="http:///www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://www.w3.org/2001/XMLSchem
a

http://www.w3.org/2001/XMLSchema.xsd">

 <xsd:element name="AddressBook"
type="AddressBookType"/>

 <xsd:complexType name="AddressBookType">
  <xsd:sequence>
   <xsd:element name="FName"  type="xsd:string"
                minOccurs="1"
maxOccurs="unbounded"/>
   <xsd:element name="LName" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

This is the data model that is generated from Example 2.

```
;;; (type = SOURCE)
;;; (access = OTXMLS.acc)
DECLARATIONS {
INCLUDE "OTMainInit.inc"
INCLUDE "OTXML.inc"
INCLUDE "User.inc"
}
Initialization {
[]
```

```
VAR->OTMdlVer = "StdVer 3.2 - Do Not Remove This
Line"
PERFORM("OTSrcInit",
"STD|XML|(null)~AddressBook~(null)", "XML")
ON_ERROR("xmlOnError")
[]
SET_FIRST_DELIM(62)   ; >
SET_SECOND_DELIM(60)  ; <
SET_THIRD_DELIM(47)   ; /
SET_FOURTH_DELIM(34)  ; "
SET_FIFTH_DELIM(39)   ; '
SET_RELEASE(92)       ; \
}*1 .. 1 ;;|-- end Initialization --|

AddressBookElement { AttributeElement "AddressBook"
   SeqGroup {
      FNameElement { Element "FName"
         FName_fld { PCData @1 .. 4092 none
         }*1 .. 1 ;; |-- end FName_fld --|
      }*1 .. * ;; |-- end FNameElement --|
      LNameElement { Element "LName"
         LName_fld { PCData @1 .. 4092 none
         }*1 .. 1 ;; |-- end LName_fld --|
      }*1 .. 1 ;; |-- end LNameElement --|
   }*1 .. 1 ;; |-- end SeqGroup --|
}*1 .. 1 ;; |-- end AddressBookElement --|
```

Data Model Generator Case 3

In this instance, only the FName or the LName is to exist. This is accomplished by defining a "choice" grouping in the XSD syntax. When the data model is generated, a RETURN rule is inserted in the PRESENT mode rules of all item within the "choice" grouping.

Once one of the items is parsed, then the RETURN is executed, since there is no need to continue to look for the balance of the siblings.

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns="po0b"
           targetNamespace="po0b"

xmlns:xsi="http:///www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.w3.org/2001/XMLSchema
```

```
http://www.w3.org/2001/XMLSchema.xsd">

 <xsd:element name="AddressBook"
type="AddressBookType"/>

 <xsd:complexType name="AddressBookType">
  <xsd:choice>
   <xsd:element name="FName" type="xsd:string"/>
   <xsd:element name="LName" type="xsd:string"/>
  </xsd:choice>
 </xsd:complexType>

</xsd:schema>
```

This is the data model that is generated from Example 3.

```
;;; (type = SOURCE)
;;; (access = OTXMLS.acc)
DECLARATIONS {
INCLUDE "OTMainInit.inc"
INCLUDE "OTXML.inc"
INCLUDE "User.inc"
}
Initialization {
[]
VAR->OTMdlVer = "StdVer 3.2 - Do Not Remove This
Line"
PERFORM("OTSrcInit",
"STD|XML|(null)~AddressBook~(null)", "XML")
ON_ERROR("xmlOnError")
[]
SET_FIRST_DELIM(62)   ; >
SET_SECOND_DELIM(60)  ; <
SET_THIRD_DELIM(47)   ; /
SET_FOURTH_DELIM(34)  ; "
SET_FIFTH_DELIM(39)   ; '
SET_RELEASE(92)       ; \
}*1 .. 1 ;;|-- end Initialization --|

AddressBookElement { AttributeElement "AddressBook"
   ChoiceGroup {
      FNameElement { Element "FName"
         FName_fld { PCData @1 .. 4092 none
         }*1 .. 1 ;; |-- end FName_fld --|
         []
            RETURN
      }*0 .. 1 ;; |-- end FNameElement --|
      LNameElement { Element "LName"
         LName_fld { PCData @1 .. 4092 none
         }*1 .. 1 ;; |-- end LName_fld --|
         []
```

```
                RETURN
          }*0 .. 1 ;; |-- end LNameElement --|
      }*0 .. 1 ;; |-- end ChoiceGroup --|
  }*1 .. 1 ;; |-- end AddressBookElement --|
```

Data Model Generator Case 4

Here, the FName or the LName can come in any order. This is accomplished by defining an "all" grouping in the XSD syntax. When the data model is generated, a repeat of the AllGroup is defined whose maximum occurrence is the number of items within the AllGroup.

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="po0c"
            targetNamespace="po0c"

xmlns:xsi="http:///www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.w3.org/2001/XMLSchema

http://www.w3.org/2001/XMLSchema.xsd">

 <xsd:element name="AddressBook"
type="AddressBookType"/>

 <xsd:complexType name="AddressBookType">
  <xsd:all>
   <xsd:element name="FName" type="xsd:string"/>
   <xsd:element name="LName" type="xsd:string"/>
  </xsd:all>
 </xsd:complexType>

</xsd:schema>
```

This is the data model that is generated from Example 4.

```
;;; (type = SOURCE)
;;; (access = OTXMLS.acc)
DECLARATIONS {
INCLUDE "OTMainInit.inc"
INCLUDE "OTXML.inc"
INCLUDE "User.inc"
}
Initialization {
[]
```

```
VAR->OTMdlVer = "StdVer 3.2 - Do Not Remove This
Line"
PERFORM("OTSrcInit",
"STD|XML|(null)~AddressBook~(null)", "XML")
ON_ERROR("xmlOnError")
[]
SET_FIRST_DELIM(62)   ; >
SET_SECOND_DELIM(60)  ; <
SET_THIRD_DELIM(47)   ; /
SET_FOURTH_DELIM(34)  ; "
SET_FIFTH_DELIM(39)   ; '
SET_RELEASE(92)       ; \
}*1 .. 1 ;;|-- end Initialization --|

AddressBookElement { AttributeElement "AddressBook"
   AllMainGroup {
      AllGroup {
         FNameElement { Element "FName"
            FName_fld { PCData @1 .. 4092 none
            }*1 .. 1 ;; |-- end FName_fld --|
         }*0 .. 1 ;; |-- end FNameElement --|
         LNameElement { Element "LName"
            LName_fld { PCData @1 .. 4092 none
            }*1 .. 1 ;; |-- end LName_fld --|
         }*0 .. 2 ;; |-- end AllGroup --|
   }*1 .. 1 ;; |-- end AllMainGroup --|
}*1 .. 1 ;; |-- end AddressBookElement --|
```

Data Model Generator Case 5

In this XSD example, two attributes were defined on the element FName. Within the generated data model an attribute initialization group (FNameListInit) is created initializing variables to either a null or default/fixed value.

A grouping (attribute_list) is then placed around the attributes with a maximum occurrence equal to the number of possible attributes to be processed. This allows the attributes to be processed in any order. Within the PRESENT rules of this group, the variables can then be referenced for their values.

It is necessary to use variables to capture the attribute's values, otherwise with each repeat of the attribute_list, the values associated with the attribute fields would be cleared.

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="po0d"
            targetNamespace="po0d"
```

```
        xmlns:xsi="http:///www.w3.org/2001/XMLSchema-instance"

        xsi:schemaLocation="http://www.w3.org/2001/XMLSchema

        http://www.w3.org/2001/XMLSchema.xsd">

<xsd:element name="AddressBook" type="AddressBookType"/>

     <xsd:complexType name="AddressBookType">
      <xsd:sequence>
       <xsd:element name="FName">
        <xsd:complexType>
         <xsd:simpleContent>
          <xsd:extension base="xsd:string">
           <xsd:attribute name="gender" type="xsd:string"
default="male"/>
            <xsd:attribute name="status"
type="xsd:string"/>
          </xsd:extension>
         </xsd:simpleContent>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="LName" type="xsd:string"/>
      </xsd:sequence>
     </xsd:complexType>

     </xsd:schema>
```

This is the data model that is generated from Example 5.

```
;;; (type = SOURCE)
;;; (access = OTXMLS.acc)
DECLARATIONS {
INCLUDE "OTMainInit.inc"
INCLUDE "OTXML.inc"
INCLUDE "User.inc"
}
Initialization {
[]
VAR->OTMdlVer = "StdVer 3.2 - Do Not Remove This
Line"
PERFORM("OTSrcInit",
"STD|XML|(null)~AddressBook~(null)", "XML")
ON_ERROR("xmlOnError")
[]
SET_FIRST_DELIM(62)  ; >
SET_SECOND_DELIM(60) ; <
SET_THIRD_DELIM(47)  ; /
SET_FOURTH_DELIM(34) ; "
```

```
SET_FIFTH_DELIM(39)   ; '
SET_RELEASE(92)       ; \
}*1 .. 1 ;;|-- end Initialization --|

AddressBookElement { AttributeElement "AddressBook"
   SeqGroup {
      FNameElement { AttributeElement "FName"
         FNameList { AttributeList
            FNameListInit {
               []
                  VAR->gender = "male" ;
default/fixed value
                  VAR->status = "" ; default/fixed
value
            }*1 .. 1 ;; |-- end FNameListInit --|
            attribute_list {
               gender_00001 { Attribute "gender"
                  gender_00001_fld { AttributeVal
@1 .. 4092 none
                     []
                        VAR->gender =
gender_00001_fld
                     }*0 .. 1 ;; |-- end
gender_00001_fld --|
               }*0 .. 1 ;; |-- end gender_00001 --|
               status_00001 { Attribute "status"
                  status_00001_fld { AttributeVal
@1 .. 4092 none
                     []
                        VAR->status =
status_00001_fld
                     }*0 .. 1 ;; |-- end
status_00001_fld --|
               }*0 .. 1 ;; |-- end status_00001 --|
            }*0 .. 2 ;; |-- end attribute_list --|
            []
               ;; assign VAR->'s here
               ;;          = VAR->gender
               ;;          = VAR->status
         }*0 .. 1 ;; |-- end FNameList --|
         FName_fld { PCData @1 .. 4092 none
         }*1 .. 1 ;; |-- end FName_fld --|
      }*1 .. 1 ;; |-- end FNameElement --|
      LNameElement { Element "LName"
         LName_fld { PCData @1 .. 4092 none
         }*1 .. 1 ;; |-- end LName_fld --|
      }*1 .. 1 ;; |-- end LNameElement --|
   }*1 .. 1 ;; |-- end SeqGroup --|
}*1 .. 1 ;; |-- end AddressBookElement --|
```

Deploying XML Data Models

Once you have completed the source and target data models, the map component files, and the development testing, you are ready to move your XML electronic commerce application to a test area or production area.

Use the procedures found in the Workbench on-line Help system, "How to Deploy."

Section 3
Testing and Debugging XML Schema Data Models

To test whether a generated data model is correct, you must process an instance of a fully defined XML document. The instance of the XML document would contain all possible elements, attributes, etc. contained in the data model.

If the XML document is inbound (that is, a source data model), run a translation to verify that the entire document has been parsed correctly by the translator. Likewise, if the XML document is outbound (that is, a target data model), then the output of a translation (for example, an instance of an XML document) may be processed by the parser and validated against the XSD.

Testing Translations

User the otrun.exe (Windows) or inittrans (UNIX) commands to run testing and production translations. Information about invoking translations can be found in the Workbench on-line help.

XSD Validation Parameters

The -V and -v arguments are used to cause XSD validations to be performed by the parser/validator. These validations check for items beyond those that indicate the document is well-formed.

Schema files (.xsd) are checked to be well-formed XML but are not validated. They are expected to be syntactically correct. Only the instance document is validated beyond checking for being well-formed.

Setting Validation Parameters

Generated data models, standard data models, and the inbound and outbound examples shown in Section 4 do not automatically validate XML data against an XSD. To validate the XML data

against an XSD, parameters must be set in the User.inc file or at the command line. The User.inc file is an Include file.

The XML Schema Plug-In uses the-DXML_VALIDATE parameter to control validation of the XML data.

The -DXML_VALIDATE parameter can be set to "Always", "WithReference", or "No".

Validating Against an XSD

XSD validation can be set globally or by a specific translation. Refer to "List of Items Validated" for additional information.

Note: These methods to control validation are necessary only when the OTCallParser.att map component file is used.

➢ **To set XSD validation globally**

1. Open the User.inc Include file using an on-line text editor.
2. Locate the DECLARE US_XMLValidate() statement.
3. Set the variable VAR->OTXMLValidate="<value>" where "<value>" can be:

Always	The translation will either validate the data or report an error.
WithReference	The translation will validate only when the DTD or XSD is referenced within the instance document.
No	The translation will not validate.

4. Save the User.inc file.
5. Close the text editor.

➢ **To set XSD validation for a specific translation**

Add the -DXML_VALIDATE=WithReference argument to the command line. For example, to set XSD validation:

UNIX Operating system

From the command line, type:

```
inittrans -at OTXMLSOxsd.att
-cs $OT_QUEUEID -DINPUT_FILENAME=OTXMLSOxsd
-DXML_VALIDATE=WithReference -tl 1023 -I
```

Windows Operating system

From the Run dialog box, type:

```
<path>otrun.exe -at OTXMLSOxsd.att
-cs %OT_QUEUEID%
-DINPUT_FILENAME=OTXMLSOxsd.txt
-DXML_VALIDATE=WithReference -tl 1023 -I
```

List of Items Validated

This list shows the items checked by the parser/validator during validation.

- Elements must exist in the namespace in which they are referenced.

  Attributes must exist in the namespace in which they are referenced.

  Validates use of the namespace attribute.

  The use of any of the 19 primitive and 25 derived data types are in compliance with their definition. The validation takes into consideration the extension and reduction using any of the 12 facets.

  Validates the 'choice' grouping.

  Validates the 'all' grouping.

  Validates the 'union' definition of elements and attributes.

  Validates the 'list' element values. Each item in the list must conform to all defined restrictions.

  Validates empty content elements.

  Validates simple content elements.

  Validates 'complexType' when defined with an attribute of mixed="true".

  Validates the use of the keywords 'INCLUDE', 'IMPORT', and 'REDEFINE' when loading schemas from multiple files.

Validates the use of the 'substitution Group' attribute to allow other elements to be used in place of another element.

Validates the use of the attributes 'any' and 'anyAttribute' and namespace.

Allows for the use of the same label name in an instance document for an element, data type, and attribute without a conflict.

Validates the use of 'documentation', 'annotation', and 'appinfo'.

Validates the use of processContent="skip" to turn off validation for the specified areas of the schema.

Data Model Validations

These items are validated when a data model is created using the XSD data model generator.

Structure
Occurrence constraints
Data types, such as, numeric, date, PCDATA
'length'
'minLength'
'maxLength'
totalDigits
fractionDigits

Additional Data Model Validations

This is a list of validations that occur by a data model created from the XSD data model generator when the -V argument is used when generating the data model. Rules are placed on the data model items for this validation.

The ID Code List is verified for schema defined enumerations.

Boolean value of 'true' ,'false', '0', or '1' only.
'Token' through the use of ID verification.
nonPositiveInteger
nonNegativeInteger
PositiveInteger
NegativeInteger
UnsignedLong
UnsignedInt
UnsignedShort
UnsignedByte
long

int
short
byte
gYearMonth
maxInclusive
maxExclusive
minInclusive
minExclusive
enumeration

List of Items Not Implemented

Refer to Appendix B, "Unsupported Items for XML Schema Version 1.0."

XML Schema Troubleshooting

Parser Error Handling

When the parser is successful, a return code of zero is generated. When an error is encountered, a non-zero error code is returned by the otxmlcanon utility.

The parser writes the error message to the standard output in the following format.

```
<PARSER_ERROR><error code>:<error message
[;detailed error message]> </PARSER_ERROR>
```

Where

Message Part	Description
<error code>	This is an integer indicating the error code.
<error message[;detailed error message]>	The error message is a short generic description of the error. The detailed error message occurs when the parser has collected additional information about the error.

Specific information about parser messages can be found in the *XML Plug-In Version 1.1.3 User's Guide.*

Types of Errors

Errors can be categorized into two groups - non-structural errors and structural errors.

Non-structural errors occur when an element or attribute in the instance document does not meet the minimum occurrence value, maximum occurrence fails or, facet verification fails. Non-structural errors are accumulated until a structural error is encountered or end of the instance message occurs, at which time all errors accumulated are output.

Structural errors occur when an unknown item or an item in the wrong position of the instance document is detected. With structural errors, processing of the instance document is stopped and all errors (both structural and non-structural) that have accumulated are output. Once the errors are written out, processing stops with either a non-zero status or a return code.

Data Model Generator Error Handling

Errors are displayed in the status area of the Workbench dialog box. Any non-zero return code represents an error.

If an error occurs when generating a data model and part of the data model is written out, the output is written to <output_filename>.log. This log file can then be opened using Workbench to see how much of the data model was generated and to find out approximately where the error occurred.

This list shows the reasons that the data model generator could fail and the resolutions for the errors.

Cause	Resolution
The specified XML document does not exist in the working directory.	Verify that the filename specified for the XML document (the Import filename) is correct and that it resides in the working directory.
The specified XML document does not refer to an existing XSD.	Verify that the filename specified for the XSD is correct and that it resides in the working directory or exists in the directory indicated in the path specified.

Cause	Resolution
Root elements in the XML document and the XSD do not match.	Verify that the root element specified is correct and matches between the instance document and the XSD schema.

The data model generator utility will send back the following return codes. Any return code that is not a zero value indicates an error.

Return Code	Description
zero (0)	Process was successful.
1	The XSD did not parse successfully.
2	File not found.

Inbound Processing, Method 2 Error Codes

When the second inbound processing method is used, the following return codes may be returned. The second method uses a pre-translation environment that parses and validates the data before translation begins. Refer to Section 1, "Inbound Processing, Method 2" for additional information.

Error Number	Description
160	INPUT_FILENAME was not supplied when invoking the translation or the referenced file does not exist.
350	The XML data is not well-formed or there was an error during validation. The original input still exists and a <sessionNo>.err file has been created. This file contains the input XML data up to the determined error and ends with a description of the error.
137	MESSAGE was not supplied when invoking the translation, the referenced map component file does not exist, or there is a syntax error in the referenced map component file.

Troubleshooting Complex XML Schema Documents

XML messages (business documents written in XML Schema syntax) provided by the customer have become increasingly large, and complex.

The following results occur frequently and are typically the root cause of some XML Schema plug-in problems. Repeated use of **otxsdgen (the XSD to data model generator),** and **otxmlcanon (the XML parser)** can detect errors and direct you to a solution. The recommendation is to use the two programs from the command line for maximum effectiveness.

Result	Action	Comment
No output is generated	Run **otxsdgen [options]** from the command-line.	An XML document must accompany the XSD. If the user creates a simple XML document of a single root element manually, then, this file could be susceptible to many minor problems.
		If the root element has not been correctly specified, then an **"Could not find root element!"** error will be generated by the model generator.
		If the XSD has been misspelled in the XML document, then a **"File error"** will be generated.

Result	Action	Comment
		If the output filename is not specified, then an **"Character/encoding...bad output"** error will be generated.
XML input file does not translate.	Run **otxmlcanon -V <file>** from the command-line.	Verify the XML document is valid. The customer may have provided an invalid XML input file. Modify the data to conform to the XSD, or ask the customer to provide a valid XML input file.

Section 4
XML Schema Examples

This chapter provides three examples for XML Schema processing. The first two example are inbound examples and the third is an outbound example.

Example 1: Inbound Processing

Inbound Example 1

Translation occurs with the use of a single environment.

➢ **To run inbound XSD translation example 1**

UNIX operating system

From the command line, type:

```
inittrans -at OTXMLSIxsd.att -cs $OT_QUEUEID
-DINPUT_FILENAME=OTXMLSIxsd.txt -tl 1023 -I
```

Windows operating system

From the Run dialog box, type:

```
<path>otrun.exe -at OTXMLSIxsd.att
-cs %OT_QUEUEID% -DINPUT_FILENAME=OTXMLSIxsd.txt
-tl 1023 -I
```

Hint: A backup file is provided since the INPUT_FILE is automatically removed at the end of a translation session. Obtain additional copies of OTXMLSIxsd.txt from the OTXMLSIxsd.tx1 file.

Inbound Example 2

In this example, translation occurs using multiple environments, thereby allowing the XML data to be completely parsed and validated before any translator parsing occurs.

➢ **To run inbound XSD translation example 2**

UNIX operating system

From the command line, type:

```
inittrans -at OTXMLSIPre.att -cs $OT_QUEUEID
-DINPUT_FILENAME=OTXMLSIxsd.txt
-DEMSSAGE=OTXMLSI2xsd.att -tl 1023 -I
```

Windows operating system

From the Run dialog box, type:

```
<path>otrun.exe -at OTXMLSIPre.att
-cs %OT_QUEUEID% -DINPUT_FILENAME=OTXMLSIxsd.txt
-DMESSAGE=OTXMLSI2xsd.att -tl 1023 -I
```

Hint: A backup file is provided since the INPUT_FILE is automatically removed at the end of a translation session. Obtain additional copies of OTXMLSIxsd.txt from the OTXMLSIxsd.tx1 file.

Example 2: Outbound Processing

Outbound Example

➢ **To run outbound XSD translation**

UNIX operating system

From the command line, type:

```
inittrans -at OTXMLSOxsd.att -cs $OT_QUEUEID
-DINPUT_FILENAME=OTXMLSOxsd -tl 1023 -I
```

Windows operating system

From the Run dialog box, type:

```
<path>otrun.exe -at OTXMLSOxsd.att
-cs %OT_QUEUEID% -DINPUT_FILENAME=OTXMLSOxsd.txt
-tl 1023 -I
```

Hint: A backup file is provided since the INPUT_FILE is removed automatically at the end of a translation session. Obtain additional copies of OTXMLSOxsd.txt from the OTXMLSOxsd.tx1 file.

Appendix A
XML Schema Plug-In Version 1.0 Implementation Files

This appendix provides a list of the files included with the XML Schema Plug-In Version 1.0.

Caution: Unless specifically instructed in the technical documentation for this product, do not change any system files installed with this product. If a system file is changed, your translations may not run properly.

Modifying system files may result in these types of problems:

- **Hindered ability to upgrade to future releases.**
- **Obstructed ability to install and run maintenance and patch releases.**
- **Memory handling issues.**
- **Degraded performance (speed) of execution.**

Shown here are examples of system filenames:

cbl*	**dummy.***	**sef***
cftest.c	**inittrans.***	**shims***
cservr.*	**uixmrt.***	**tmbench.***
dfapi.*	**otrans.***	**tradegui.***
dfshl.sh	**rms***	**uixm.***

Any filename beginning with "OT" or "ot"

XML Schema Files Provided

The following list describes map component, access, and data model files included with the XML Schema Plug-In Version1.0.

Filename	Description
datatypes.dtd	List of schema data types.
libSchema.1.sl	Hewlett Packard schema validator shared library (used instead of xmlSchema.dll)
libSchema.1.so	AIX, DEC, and SUN schema validator shared library (used instead of xmlSchema.dll)
OTXMLDataType.tbl	Table used by the XSD data model generator which cross-references the schema data types with the product's access model items.
otxsdgen.exe	Executable which creates a data model structure from the XSD specified in the XML instance document.
readmeXSD.txt	A list of open issues that were discovered but not corrected before commercial release
vc6-re300dl.dll	Windows regular expression shared library.
vc6-re300readme.txt	Windows regular expression shared library readme file.
WBXMLSchema_en_US.properties	Workbench Import dialog box for the XSD model generator.
xmlSchema.dll	Windows shared library used by the parser and schema model generator.
xmlschema.dtd	Document type definition from which the schema-for-schema is derived.
xmlSchmea.xsd	Schema-for-schema from which all other schema's are derived.
xsdp_cat.txt	Error message catalog file used for localization
xsdp_EN_USA.cat	English localization catalog for error messages.

The following files are used in the examples included in this manual:

Filename	Description

Filename	Description
OTXMLSI2xsd.att	Map component file used for inbound XSD processing, second approach.
OTXMLSIunixxsd.att	UNIX version of OTXMLSIxsd.att
OTXMLSISxsd.mdl	Sample inbound XSD source data model. Used to parse OTXMLSIxsd.txt
OTXMLSIxsd.att	Inbound map component file defining source and target models to parse XML XSD data and create the UDF. It is used by OTXMLSIxsd.bat and OTXMLSIxsd.sh.
OTXMLSIxsd.bat	Windows batch file used to invoke the sample XSD inbound XML translation.
OTXMLSIxsd.sh	UNIX shell script used to invoke sample XSD inbound XML translation.
OTXMLSIxsd.txl	Backup input data for the XSD inbound XML sample (XML file.).
OTXMLSIxsd.txt	Input data for the XSD inbound XML sample (XML file).
OTXMLSIxsd.xsd	XSD file used in the inbound samples.
OTXMLSOSxsd.mdl	Sample XSD outbound source data model used to parse OTXMLSOxsd.txt.
OTXMLSOTxsd.att	Sample XSD outbound target map component file.
OTXMLSOTxsd.mdl	Sample XSD outbound target data model used to create XML output from OTXMLSOxsd.bat and OTXMLOxsd.sh.
OTXMLSOxsd.att	Outbound map component file defining source model to parse UDF data. Used by OTXMLSOxsd.bat and OTXMLSOxsd.sh.
OTXMLSOxsd.bat	Windows batch file used to invoke XSD outbound sample XML translation.
OTXMLSOxsd.sh	UNIX shell script used to invoke the XSD outbound sample XML translation.
OTXMLSOxsd.txl	Backup input data for the XSD outbound XML sample (Application File).
OTXMLSOxsd.txt	Input data for the XSD outbound XML sample (Application File).

Appendix B: Unsupported Items for XML Schema Version 1.0

There are several differences between the W3C® XSD Schema specification and the XML Schema Version 1.0 Plug-In. Items shown in this appendix are not supported by this release of the XML Schema Version 1.0 Plug-In.

Unsupported Parser/Validator Issues

The following describes variations between the 2 May 2001 W3C® XSD schema specification and the XML XSD Plug-in implementation. This list represents items that refer to the parser's validation against the schema.

Note: The items shown here are not supported in the XML Schema Plug-In Version 1.0 implementation. They are items that are included in the W3C XSD schema specification but are not included with the plug-in software.

In the W3C® XSD schema specification, the "duration" data type states that "T" should be present only when time is being conveyed in the value. The XML XSD Plug-in implementation allows "T" to be present even when time is not being conveyed (P1Y2MT). Also a duration value of "P" is permitted, since all days and times are optional.

- Data types are being validated using regular expression patterns. Because of this, invalid days (for example, 32 days in a month), year (0), months, hours, minutes and seconds (more than 60 seconds) are not determined to be invalid. Dates are validated for proper digit values when parsed using the #DATE access function. This occurs for data types: date, gYearMonth, gYear, gMonth, gMonthDay, gDay. This does not happened for the data types of dateTime and time, since they are parsed as strings.
- Validation does not enforce use of "key" and "keyref" identity-constraint definitions. "key" allows a value to be defined as unique, cannot be set to nillable and referencable within an instance document. "keyref" is used to define a value that must reference a "key" value within the same instance document.
- Validation does not allow more than 6 digits for the year, although the W3C® XSD schema specification allows for any number of year digits.
- When the facet of length is defined in addition to minLength and/or maxLength, first both min and max are set to length, then min is set to minLength, if defined, and max is set to maxLength if defined. Validation then uses min and max.
- The data type "normalizedString" is parsed as a "string" and contrary to the W3C® XSD schema specification, it allows for tab, carriage return and the line feed characters.
  The data type "token" is parsed as a "string" and, contrary to the W3C® XSD schema specification, allows for tab, carriage return and the line feed characters, along with leading and trailing spaces, and internal sequences of two or more spaces.
- Validation applies only to the entire XML instance document. When the root element does not reference a namespace, but elements within do, these single elements or group of elements are not validated.
- The "whiteSpace" facet is not enforced. The three settings for whiteSpace are preserve, replace and collapse. With the preserve setting, no normalization is done, the value is not changed. With the replace setting, all CR, LF and tabs are replaced with space. With the collapse setting, multiple spaces are collapsed down to one and trailing spaces are removed, after performing replace.
- The first character of the value is within a certain character set is not enforced for the Name, NCName, ID, IDREF, and ENTITY data types. These data types are validated as if they were "string". They will not catch invalid leading characters such as: '-', '.', or digits.
- The XML XSD Plug-in implementation does not enforce the "unique" constraint when validating. This is when an attribute or element value must be unique within a certain scope of the instance document.The XML XSD Plug-in implementation does not enforce an abstract attribute defined on an element - abstract="true" when validating. Abstract allows for definition in the schema but not its use within the instance document.

- The processContent attribute on elements is partially implemented. "processContent="skip" (do not validate) and processContent="strict" (validate) are enforced. "processContent="lax" (only validate when schema information is available) is processed the same as "skip".
- The "long" and "unsignedLong" data types generate erroneous results when 16 or more digits are used in the value. For example, it may report a good value in error or a bad value as good.

Unsupported Model Generator Issues

The schema data model generator was developed from the XML XSD Plug-in schema parser/validator. This list identifies variances with the XSD schema model generator.

Note: The items shown here are not supported in the XML Schema Plug-In Version 1.0 implementation. They are items that are included in the W3C XSD schema specification but are not included with the plug-in software.

Data Model Generator Issues

Field lengths are defaulted when the length (length, maxLength, minLength) facets are not specified in the schema. The default is 20 for numeric, date is specific to its data type (gYear would be 4 for 4 - YYYY) and 4092 for all others.

"length", "minLength" and "maxLength" can represent lengths greater than 4092, whereas the base product's maximum length is 4092. All facet lengths defined greater than 4092 will be set to 4092 when the data model is generated.

Data types not referenced in the OTXMLDataType.tbl file default to the first entry in the file, which is "string". All schema defined Primitive and Derived data types are contained in the .tbl file.

If the facet of length is defined in addition to minLength and/or maxLength, first both minSize and maxSize of the data model item are set to length, then minSize is set to minLength if defined, and maxSize is set to maxLength if defined.

Rules for the minExclusive, minInclusive, maxExclusive and maxInclusive facets are placed into the generated data model for the byte, unsignedByte, integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort, decimal float and double data types. They are not generated for hexBinary, base64Binary or the date and time data types.

Enumerations (list of values, like ID codes) defined on the source root element are not added to the .ids file during model generation. For the generation of a target data model, they are.

Mandatory attributes on the source root element are not enforced in the data model. The are enforced during schema validation.

The data model does not default or enforce element values defined with a "default" or "fixed" attribute, when no value is present for the element in the instance document. Attributes do default values when not present and are defined with a "default" or "fixed" attribute.

Date with a year greater than 9999 is not supported. The base product allows the year to be 0000, but the W3C® specification does not allow a year value of zero.

Using TimeZone for the date and time data types (dateTime, time, date, gYearMonth, gYear, gMonth, gMonthDay, gDay) is not supported in the data model's format. For a time zone received as part of a value, the access type must be changed from DateFld to PCData with the value being handled like a string rather than a date or time value.

The W3C® specification allows dateTime, date, gYearMonth and gYear to be negative. This is not supported in the generated data model.

When a union is defined within the schema, only one item defined as PCData is created in the data model structure. It has no restrictions on it. Therefore, if the various union components have different restrictions (facets) defined on them, they are not reflected within the data model's rules for validation.

Substitution elements (elements defined with the substitutionGroup attribute) are not added to the data model when generated.

An "any" type element and an "anyAttribute" type attribute are not added to the data model when generated. A group data model item is inserted as a placeholder.

A "list" type item within the data model treats the list value as one string rather than a number of token values. Using data model rules, the list string can be reduced down to individual tokens.

The W3C® schema specification identifies that the data types of float and double should support values of NaN (not-a-number), INF (positive infinity) and -INF (negative infinity). These strings will report a 146 format error when parsed by the #NUMERIC access function.

All data model items created for signed numeric data types use the base product's sign masking character of "N". For the following signed numeric data types, the designation of a positive value can be represented with either the '+' character or the absence of any character: decimal, integer, nonNegativeInteger, positiveInteger, long, int, short and byte. If numeric values for these data types are received or need to be generated with the '+' character, the data model item format needs to be changed from using 'N', to using the '+' masking character. For example, change "NFFFFFFFF" to "+FFFFFFFF".

Translator validation for mandatory attributes (PERFORM("xmlAttrConditionalnn")) is configured to validate a maximum of twenty mandatory attributes per element only.

Mandatory components of an "all" grouping are not enforced in a source data model. For the target they are enforced, since the data model controls the sequence the items are written out.

The generated data model items for data types of "time" and "dateTime" use the access model item of PCData rather than a date or time access item. This means that no formatting is included in these data model items because the available masking characters cannot address all the complexities of the values that can be conveyed in these types of fields. If a time or date and time is known to be formatted in a consistent manner within these fields, the data model item can be changed to use date or time access items, along with a format.

The complexType data type declaration with its mixed attribute set to "true" is not supported in the generated data model. Unmixed allows text to be placed between elements and their child elements.

XML schema implementation allows for redefinition of elements within the instance document. Since the data model is generated from the schema and not the instance document, it is unable to recognize redefinition during inbound translation when it occurs.

Index